Cranfield University

Yuke Ma

Design, Test and Implement a Reflective Scheduler with Task Partitioning Support of a Grid

PhD thesis

School of Engineering

Cranfield University, School of Engineering

Department of Process and Systems Engineering

Applied Mathematics and Computing Group

PhD thesis

Academic Year 2007-2008

Yuke Ma

Design, Test and Implement a Reflective Scheduler with Task Partitioning Support of a Grid

Supervisor: Professor Chris Thompson

May 2008

The thesis is submitted in fulfilment of the requirement for the degree of Doctor of Philosophy

Abstract

How to manage a dynamic environment and how to provide task partitioning are two key concerns when developing distributed computing applications. The emergence of Grid computing environments extends these problems. Conventional resource management systems are based on a relatively static resource model and a centralized scheduler that assigns computing resources to users.

Distributed management introduces resource heterogeneity: not only the set of available resources, but even the set of resource types is constantly changing. Obviously this is unsuitable for the present Grid. In addition, the Grid provides users with the physical infrastructure to run parallel programs. Because of this increasing availability, there are more requirements for parallelization technologies.

Therefore, based on problems outlined above, this thesis provides a novel scheduler which not only enables dynamic management but also provides skeleton library to support the task partition. Dynamic management is derived from the concept of reflectiveness, which allows the Grid to perform like an efficient market with some limited government controls. To supplement the reflective mechanism, this thesis integrates a statistical forecasting approach to predict the environment of the Grid in the next period.

The task partitioning support is extended from the skeleton library in the parallel computing and cluster computing areas. The thesis shows how this idea can be applied in the Grid environment to simplify the user's programming works.

Later in this PhD thesis, a Petri-net based simulation methodology is introduced to examine the performance of the reflective scheduler. Moreover, a real testing environment is set up by using a reflective scheduler to run a geometry optimization application.

In summary, by combining knowledge from economics, statistics, mathematics and computer science, this newly invented scheduler not only provides a convenient and efficient way to parallelize users' tasks, but also significantly improves the performance of the Grid.

Key words: dynamic management, reflective, automatic partition

Acknowledgements

First and foremost, I would like to express my profound gratitude and respect to my supervisor, Professor Chris P. Thompson, for his valuable guidance, suggestions, and encouragement throughout my postgraduate study at Cranfield University. This thesis could not have been completed without his support.

I am also indebted to Professor Frank Zhigang Wang for his helpful comments, instruction on papers.

I would also like to thank my parents and my girlfriend Shuai Liu for the moral support they provided during the course of this Ph.D. Without them this work would have not been possible.

I must also mention the great help received from Rachael Wiseman and Bo Xu.

Finally, I thankfully acknowledge the financial support provided by the Applied Mathematics and Computing Group at Cranfield University during the course of this Ph.D.

Table of Contents

List of Figures

List of Tables

List of Code Fragments

Chapter 1 Introduction

Grid computing [36][38] has been regarded as the next generation of World Wide Web (WWW), which provides an infrastructure allowing users to share all types of computing resources globally. It is obvious that, on one hand there are abundant idle resources, and, on the other hand, a mass of computational problems that cannot be solved due to lack of computing power. To address this issue, industries and governments tried to develop parallel computing, cluster computing, and distributed computing technologies. However, these technologies were unable to coordinate these resources efficiently on a global scale. These restrictions led people to look for alternative solutions to integrate these resources and the Grid offered a way of dealing with the problem. However, there were still many technical challenges to solve before it becomes available to public usage. In this chapter, we first discuss the concept of the Grid and its special characteristics. After this, we put forward the difficulties we need to solve in Grid scheduling.

1.1 Evolution of Grid

People often treated I-WAY (Information Wide-Area Year) [94] as the first real example of a Grid system, which was an experimental demonstration for the project SC95. It opened the door to consideration of how the Grid system should look and behave. Early internet based distributed computation solutions such as SeTI@Home [90] (and later BOINC) exemplifies the power of federated computing. Following this, the Globus [42][44] and Legion [26] projects proposed more standard approaches to develop system-level Grid infrastructures. The Condor [45][60] project focuses on high-throughput scheduling, while the Network Weather Service [106] is concerned with resource prediction. The NetSolve [11] and Ninf [69] projects are used for remote computation. The AppLes [21], APST [24], JOSH [55] and Mars [2] projects explore high-performance scheduling. All these experiments and ideas helped to instigate a modern theory of Grid computing. However, these Grid infrastructures are incompatible with each other.

In the late 1990s, the Globus project team discovered the great potential of XML-based Web Services. They switched from Globus Toolkit development to a Web Service platform OGSA (Open Grid Service Architecture) [36][38][39][40] in order to set up Grid standard protocols. After OGSA, the interfaces to the outside Grid system were based on Grid Services. Because the technologies it applied are platform-free, such as XML (eXtensive Markup Language) [17], SOAP (Simple Object Access Protocol) [81], WSDL(Web

Services Description Language) [67], and UDDI (Universal Description, Discovery and Integration) [92], all the problems about heterogeneity were hidden.

Every year, the American government invests approximately US$ 500 million in Grid infrastructure research. The Science Grid, supported by the Department of Energy, connected its two super computers through ESNet Grid [83]. The TeraGrid [25][82] from the American National Science Foundation connects five super computers from five different locations. The Department of Defence is implementing its GIG (Global Information Grid) [18], which is planned to be completed in 2020.

In addition to the American government, there are many companies who have invested in this area. One example is IBM, which has provided US$ 4 billion for its project called Grid Computing Initiative.

The UK government also invested £100 million in its UK National Grid.

The EU started several Grid projects, such as DataGrid [47], UNICORE [35], MOL [1] etc.

1.2 Concept of Grid

Grid [36][39][43][44] is an advanced technology and infrastructure, the goal of which is to supoort all types of computing resources [39][44] into a single

management system which provides easy-access and reliable in a standard and economical manner. In addition to all kinds of computers, the computing resources also include networking, data, scientific devices, licences, backup, error-reporting and even the work of human beings.

The ideas of the Grid were brought together by Ian Foster, Carl Kesselman and Steve Tuecke, widely regarded as the "fathers of the Grid" [43][44]. They led the effort to create the Globus Toolkit [39][42] incorporating not only computational management (examples: cluster management and cycle scavenging) but also storage management, data transition, authentication and a toolkit for developing additional services such as resource discovery, resource registration and notification services.

The concept of Grid computing given above is an abstract and general definition. If we consider a more specific definition, Grid computing is a way of providing a means of helping to solve massive and complicated computational problems by coordinating free computing resources (CPU power and storage) from a group of disparate computers. The aim of the Grid is to provide the ability to manage all the computing resources across different administrative domains which cannot be completed by traditional clusters or distributed computing.

The view [110] that a Grid merely provides a way to connect computers, devices, data and personnel through networking is out-of-date, because it only emphasises the physical networking infrastructure and the discrete computing resources, not a unified entity. Another view regards the Grid as middleware

systems, which is also not accurate, as it over-emphasises the importance of software management functions. A comprehensive description, combining the physical resources and the management system, is required to describe the Grid system.

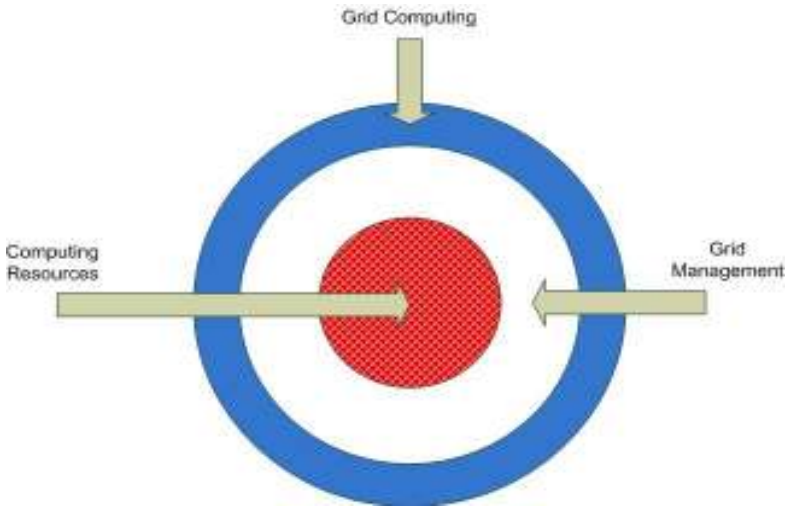

Figure 1: the relationship among Grid resources, Grid management and Grid computing

Because of its massive computational power, the Grid offers a way to solve huge calculation applications like biological modelling, financial pricing, earthquake simulation, and climate/weather forecasting. The first appearance of the term Grid computing [20][88] can be traced to the early 1990s as an analogical creation to access computer power as convenient electric power grid. The definitive criteria to judge a Grid is provided by Ian Foster in [44]. The three points on his checklist are:

- *"the ability, using a set of open standards and protocols, to gain access to*

applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across 'multiple' administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements"

- *"Grid computing reflects a conceptual framework rather than a physical resource."*
- *"The Grid approach is utilized to provision a computational task with administratively-distant resources."*

1.3 Characteristics of Grid

As the Grid is very different from the traditional computing infrastructure, it is important for us to understand its characteristics so that we can develop a more suitable management system. In this section, we will introduce some fundamental concepts to facilitate our discussion.

1.3.1 Distributed and Sharing

Distribution is one of the most important characteristics of the Grid. When we say a Grid is distributed, we mean a Grid is made up of different computers, databases, digital libraries and other scientific devices which are all located in different areas.

This property is key to the computational problems on Grids normally

distributed, not centralised. As a result of this, the management system needs to solve problems such as resource management, task scheduling, transmission security, using real-time systems and possible human intervention.

Although the resources are distributed, they can be fully shared through the Grid, which means that all the resources can be provided to any users joining the Grid. “Sharing is the purpose of the Grid and without it the Grid is meaningless” [38][110], so the provision of resource sharing is the core idea of the Grid. This is a very braod concept, which not only means the computers in one location can be used to finish a remote task, but also enables sharing the intermediate results, databases, models and so on.

Generally speaking, distribution is the physical characteristic and sharing implement the logical characteristic with the support of Grid management system.

1.3.2 Self-similar

Self-similarity [20][44] exists widely in natural and social phenomena. Most complex systems have special characteristics, like the Grid. Local parts of the Grid are similar to the global part of the Grid, which means the local part has the characteristic that exists in the global. It can be expressed in some degree by the concept of recursion.

1.3.3 Dynamic and diversified

We cannot assume the Grid’s structure is constant. The resources in the Grid

could leave the environment or become instantly faulty. Or resources outside of the Grid could join the environment as time lapses. Dynamism here means the dynamism increasing and decreasing.

The resources are heterogeneous and diversified in the Grid, therefore the management system must solve problems such as communications between different operating systems.

1.3.4 Self-manageable

The resources in a Grid firstly belong to specific organisation or person, so the owner of the resources has the highest administration rights. In the meantime, the resources in the Grid should be under the control of the Grid itself, or no connections can be set up.

1.4 Requirement for Scheduling in Grid

According to the Grid's current information and forecasting information, the scheduler maps the independent tasks into suitable resources and executes them at a specified time. It plays an important role in the computer multitasking, multiprocessing operating system and distribution system. In the Grid, the scheduler is one of the most crucial parts of the whole system. Many projects have addressed the resource selection problem. Systems such as NQE [78], PBS [53], LSF [59], and Load Leveler [103] process user submitted jobs by finding resources that have been identified either explicitly through a description control language or implicitly by submitting the job to a particular

queue that is associated with a set of resources. This manually configured queue hinders the dynamic resource discovery. Globus [42][44] and Legion [26], on the other hand, present resource management architectures that support resource discovery, dynamical resource status monitor, resource allocation, and job control. These architectures make it easy to create high-level schedulers.

Normally the tasks are executed on remote nodes, so the users have very limited control powers over these tasks. In order to provide efficient management, the Grid needs to have a different scheduling module compared to the traditional schedulers, which require a dynamic control function and the automatic partition.

- Dynamic Control

  The environment of a Grid varies from time to time in an unconstrained fashion. Therefore it is hard to predict the future condition of the Grid. Furthermore, because it is unlikely to receive accurate information, the scheduler may not make a suitable decision. In order to allow more to be done in a given amount of time and to have all users served faster and more efficiently, it is necessary to schedule the user's requests to suitable service resources and also to add dynamic controls which vary depending on the Grid architecture and usage. This creates specific management problems depending on how the user's requests are scheduled and how resources are controlled at runtime.

- Task Partitioning Support

Evidently, the Grid provides users with the physical infrastructure to run parallel programs. Because of this convenience, there is increasing requirement for parallel technologies. So how the users can be permitted to take advantage of the Grid becomes another increasing concern. If the Grid can provide a kind of service which helps the users to divide the task into sub-tasks and manage these tasks, This will significantly ease the users workload in writing parallel programs. A skeleton structure supporting commonly-used patterns of parallel computational interactions could be packaged up as parameterisable library classes (or language control structures) so that subsequent users can benefit from finely tuned implementations of the underlying structure, without "re-inventing the wheel" and without risking the introduction of complex errors. Because of the structured library, it becomes easier for the system to control the workflow even in a very complicated distributed environment.

In this thesis, we have proposed a new scheduler based on the skeleton programming structure and dynamic resource management, in order to meet the needs of providing high standard services in a rapidly changing Grid environment.

1.5 Organisation of the thesis

The rest of the thesis is organized as follows: Chapter 2 reviews popular scheduling heuristics for traditional computing systems, presents an overview

of different Grid schedulers and a comparison with other scheduling technologies. The thesis then concentrates on the design of the reflective mechanism in Chapter 3. Chapter 4 then discusses mathematical modeling, using Petri nets and its evaluation via simulation. Combining previous chapters, an overview regarding how the whole system works is presented in Chapter 5. Also detailed designs and implementations of the reflective scheduler are discussed in Chapter 5. This is followed by Chapter 6 which describes the Virtual Testbed project integrating the reflective scheduler to solve some time-consuming shape optimization calculations. Finally, the thesis concludes and presents ideas for future work in Chapter 7. In the Appendix we discuss the skeleton programming technology and test its performance in a general Grid environment comparing to cluster environments.

Chapter 2 Literature Review

Generally speaking, a Grid scheduling process is a workflow that maps and manages the execution of inter-dependent tasks on distributed resources. It assigns suitable resources to tasks in order to meet the objective requirements imposed by users. Proper scheduling can dramatically improve the performance of such systems. However, the problem of mapping tasks onto distributed services belongs to a class of problems known as NP-hard problems [102]. In other words, there is no algorithm to find the optimal solution in polynomial time. Even though the scheduling problem can be solved by using brute-force search, which is a trivial but very general problem-solving technique, that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem statement. However the complexity of methods for solving it is very high. However in Grid environments, scheduling decisions must be made in the shortest time possible, because there are many users competing for resources, and time slots desired by one user could be taken by another user at any moment. Many heuristics and meta-heuristic-based algorithms have

been proposed for schedule applications in heterogeneous distributed system environments. In this chapter, we will discuss several existing scheduling algorithms developed and deployed in various Grid environments.

2.1 Basic Scheduling Heuristics

In this section, we firstly review a set of heuristic algorithms which schedule meta-tasks to heterogeneous computing systems. A meta-task can be defined as a collection of independent tasks with no data dependencies. In traditional stable computing environments with very limited or no changes in infrastructure, meta-tasks can be mapped onto machines statically. Therefore each machine executes a single task at a time.

A large number of heuristics [6] have been designed to schedule tasks to machines on heterogeneous computing systems. Eleven commonly used heuristics, UDA [79][80], Fast Greedy [79], Min-min [56][79][80], Max-min [56][79][80], Sufferage [63], XSufferage [23], GA [57][109], SA [71][103], GSA[51], Tabu [58][80] and A* [61] are described follows.

UDA (User-Directed Assignment) [79][80]: mapping each task in arbitrary order to the computing resources with the shortest expected starting time. The user does not need to know the condition of that resource. In this case, the scheduler only helps to send the task to the assigned resources. When to execute the task will totally depend on the resource itself. If the resource is

lightly loaded or idle, the task can be executed immediately. Alternatively, the task will be executed whenever the resource becomes available again. This algorithm can be easily implemented, however there is a critical disadvantage, as the completion time is totally random.

Fast Greedy (also called Minimum Completion Time) [79]: mapping every task in arbitrary order to computing resources so as to deliver the shortest expected completion time without considering the minimum execution time. In this case, as this heuristic may result in the task to be executed for longer, it may cause the user larger costs.

Min-min [56][79][80]: the heuristic first constructs a task pool and then computes the completion time for each task. The task with the minimum completion time is mapped to the specified computing resource. The corresponding task is deleted from the pool, and then the procedure will be repeated until no tasks remain. Min-min schedules the “best case” tasks first and generates relatively good schedules. Generally speaking, the Min-min heuristic not only provides a simple and fast which is also stable. However, the drawback of Min-min is that it assigns the smallest task first and then a few larger tasks execute while several machines sit idle, resulting in poor machine utilization.

Max-min [56][79][80]: The Max-min heuristic is very similar to the Min-min algorithm. The difference is that the task with the maximum completion time is mapped to the specified computing resource.

Sufferage [63]: The Sufferage heuristic assumes that if the task is not mapped to this resource, the system will suffer the biggest loss. In this heuristic, each task has a Sufferage value to define the gap between the best completion time and the second best completion time. The higher the Sufferage value the task has, the higher its priority. The heuristic can be described as:

- For each task determine the difference between its minimum and second minimum completion time over all machines (Sufferage value)
- Over all tasks find the maximum Sufferage value
- Assign the task to the machine that gives this Sufferage value
- Iterate till all the tasks are scheduled

XSufferage [23]: The XSufferage heuristic is derived from the Sufferage. This new version is used especially in the Grid environment. The Sufferage value is computed not with one single computing resource, but with several different resources.

GA (Genetic algorithm) [57][109]: is a widely used heuristic to solve optimisation problems. This heuristic assumes that the potential solution can be represented by a combination of a population of parameters, chromosomes in this case. There are two ways of generating the chromosomes, using the outputs from other heuristics, or randomly selecting a certain number of chromosomes. From the existing chromosomes, the algorithm will combine them in a different order to generate new ones. After

considerable iterations, the new chromosome pattern becomes stable and can be regarded as the optimised solution to the problem.

SA (Simulated Annealing) [71][103]: is also a popular solution to optimisation combination problems. It is named from analogy with the annealing procedure in mechanical engineering. The biggest advantage of this algorithm is that it can find the global solution. At the start, the “temperature” is higher, so the algorithm can accept some relatively bad possible solutions in order to find a better solution in the search space. As the searching proceeds, the “temperature” decreases, and worse solutions are eliminated. Finally, the output of the best solution is found. The total execution time can be used as the temperature in this case.

GSA (Genetic Simulated Annealing) [51] is a combination of GA and SA. Generally speaking, this approach is more like the Genetic algorithm. The only difference is that the selection criteria of the chromosomes are decided by the Simulated Annealing.

Tabu [58][80]: Tabu uses solution space search methodology. The difference is that it will record all the space it has searched before in its search list. It can thus avoid searching spaces similar to previous ones. However, the concept of the Tabu is sometimes too strict, and needs other heuristics to assist in its search list management.

A* [61]: A* is another classic solution space search heuristic, and can be found in a lot of the task scheduling problems. This heuristic is based on tree searching. The root is a null node and the intermediate nodes represent partial solutions. The leaves of the tree represent the complete solutions. Each node has a cost function to represent the lower bound of the partially optimised solution. The search starts from the root. The node with the minimum cost function will be replaced by its children. The search will end when the mapping reaches to a leaf.

Research [56][63] has showed that UDA, Max-min, SA, GSA, and Tabu do not, in general, produce good schedules. Min-min, GA, and A* are able to deliver good performance in the static computing environments. The difference between the completion times of the schedules generated by these three algorithms is less than 10%. GA is consistently better than Min-min by a few percent, since it is “seeding” the population with a Min-min chromosome. A*, on the other hand, produces better or worse schedules than Min-min and GA in different situations. Among the three algorithms, Min-min is the fastest algorithm, GA is much slower, and A* is very slow. Even GA has to “seed” the population with a Min-min chromosome to obtain a good performance. It is worth noting that these comparisons are made in a static environment not a dynamic one. Also in those heuristics, the task’s expected computing time should be known in advance. Lots of the one-Step-Ahead [21] load prediction

approaches have been developed in order to understand the task's expected computing time.

2.2 Some Popular Grid Schedulers

In the above section, we have talked about the basic scheduling heuristics for heterogeneous computing. These basic heuristics have now been applied to a Grid environment in order to develop Grid schedulers. The Globus Toolkits 2.x [42][44], 3.x [39] and even Globus Toolkit 4.x [44] do not provide a job scheduler or meta-scheduler. However, there are a number of job schedulers available that already are, or can be, integrated with Globus. Schedulers usually react to the immediate Grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower-level schedulers rather than to an individual machine. They can also work on a peer-to-peer basis.

For solving related issues of scheduling and load in distributed environments, there exist a number of strategies and approaches. Nimrod [4][19] employs a parametric engine and heuristic algorithms for scheduling tasks. Condor and LSF [59] address resource management within a local Grid. The local Grid scheduler described in this work is based on application performance prediction. AppLes and Ninf [69] are based on performance evaluation

techniques to give a good scheduling for balancing the workload. Agent-based resource discovery is also used, where each agent either represents a user application, a resource, or a matchmaking service. Rather than using a collection of predefined scheduling schemes, this work uses a reflective scheduler that can reconfigure itself with different roles at run time. To help the performance prediction toolkit PACE [47] utilizes a genetic algorithm to schedule local tasks and apply agent-based technology to balance the high-level workload. But they do not focus on questions of widespread load balancing for Grid service. In our research we introduced dynamic scheduling to solve the load balancing constraints. This is an important difference from the above work, which facilitates more effective job dispatch.

Examples of such systems are Nimrod/G [4][19], Condor-G [45][60], APST [21][24] and EU-DataGrid Broker [10] (later succeeded by gLite). These are chosen for detailed comparison against the reflective scheduler as their objectives and approaches are similar to that of the scheduler.

Nowadays, there are many resource management and scheduling systems being developed. All of them have their own advantages and disadvantages.

Globus defines the architecture for resource management of autonomous distributed systems with provisions for policy extension and co-allocation. Customers describe their required resources through a resource specification language (RSL) [36][39] that is based on a pre-defined schema of the resources database. The task of mapping specifications to actual resources is

performed by a resource co-allocator, which is responsible for coordinating the allocation and management of resources at multiple sites. Using RSL, customers may provide very sophisticated resource requirements, but servers have no analogous mechanism.

The JOSH[55] gives a method of calculating the performance of different resources, but there are no methods available to evaluate the performance of different tasks. So to provide a method to evaluate both the performance of the resource and the task will produce a more powerful matchmaking tool for the scheduler.

Systems such as NQE [78], PBS [53], LSF[59] and Load-Leveler [103] process jobs by finding resources that have been identified either explicitly through a job control language, or implicitly by submitting the job to a particular queue that is associated with a set of resources. Customers of the system have to identify a specific queue to submit to a priority, which then fixes the set of resources that may be used, and hinders dynamic qualitative resource discovery. Furthermore, system administrators have to anticipate services that will be requested by customers and set up queues to provide these services. Over time, the system may accumulate a large number of queues whose service semantics differ to various extents, complicating the process of finding the appropriate queue for a job.

Legion [26] takes an object-oriented approach to resource management, formulating the matching problem as an object placement problem. The

identification of a candidate resource is performed by an object mapper[26], whose recommendation is then implemented by a different object. The Legion system defines a notation that is similar to classads (classified advertisements)[45][60], although it uses an object oriented type system with inheritance to define resources, in contrast to the simple attribute-oriented Boolean logic of classads. Legion supports autonomy with a jurisdiction magistrate (JM) [105], which may reject requests if those offered do not match the policy of the site being managed by the JM. While the JM has resource veto power, there is no way for a resource to describe those requests that it would rather serve.

Distributed computing environments such as Seti@Home[90] and Distributed.net[48] exemplify the power of federated computing. However, these systems do not provide general and flexible mechanisms to specify resource usage and access policy, running tasks in “screen saver” priority instead. However, this policy may neither be necessary nor sufficient to many resource owners. Furthermore, the infrastructure to match customers to resources is also rudimentary when compared to the matchmaking system.

The JINI system being developed by Sun Microsystems has notions similar to the classad based Condor system: resources advertise their presence, customers discover their presence through a lookup service and claim them for computation. The JINI architecture is closely coupled to the Java platform, and the lookup service used by customers essentially locates object instances

that implement the interface specified by the customer. Constraint based queries may also be specified by the customer, but the query language is significantly less rich than the classad language.

None of the resource management and scheduling mechanisms mentioned above, provide a method to allow the scheduler to adjust its own action or own resource management and scheduling mechanism according to the fast-varying Grid environment.

2.2.1 Condor and Condor-G

As we know, the Condor's matchmaking is a successful example of managing the tasks in a single administrative domain. Within a single domain, Condor can provide a high-throughput scheduler for the local non-dedicated computing resources. It also provides the task recovery mechanisms such as checkpoint and migration. Furthermore, Condor supports the remote procedure call (RPC) function, which allows a local machine to accept the remote call request.

Globus is designed to manage tasks across multi-administrative domains. The Globus Security Infrastructure(GSI)[41] provides authentication checks across different domains. The Globus Resource Allocation Management (GRAM)[30] and Dynamic-Updated Request Online Co-allocator (DUROC)[33][42] can manage the scheduling problem in multidomain and heterogeneous computing resources. The Globus Access to Secondary Storage (GASS)[15] and GridFTP[7][8] allow the user to access the data from different locations.

The Globus Resource Information Service (GRIS)[30][42] and Globus Index Information Service (GIIS)[30][42] provide the resource registration and discovery functions.

Condor can provide a very simple but stable management system in a single administrative domain, and Globus is good at providing a unified system by combining heterogeneous computing resources, Condor-G is designed to combine the strengths of Globus and Condor systems to manage the tasks in the Grid. Condor-G can manage the tasks with its full-featured queuing service, credential management and fault-tolerance.

If we want to consider fault tolerance, then we have to include two types of problems: the local computing resources have crashed and the networking connection is down. When the resources have crashed, the Condor-G can first store the queue information on the disk and then re-send the task to a machine to execute. If the network is down, the system will wait until connectivity is restored and then re-send the task to the machine to execute.

The authentication in Globus is by using time limited X509 proxies, therefore Condor-G provides credential management to refresh the proxy or create the new proxy to execute the tasks.

Queuing management [60] contains maintaining a persistent queue, providing queue-manipulation tools, creating the task dependencies, and accessing the log files.

The GridManager [45][60] is used to help submit the tasks to the remote Globus system. We can see that Condor's matchmaking [45][60] mechanism is a very good idea for us to use, because matchmaking services enable discovery and exchange of goods and services in marketplaces. Agents that provide or require services advertise their presence by publishing constraints and preferences on the entities they would like to be matched with, as well as their own characteristics. A matchmaker uses a matching operation to discover pairings between compatible agents. However, there is no numerical representation for this match, in other words, we cannot judge which match is better. So in the new design, the system will give different values for different matches. Thus matches can be compared and analyzed numerically.

Condor-G supports different models such as workflow-based and also different customised scheduling strategies. However, it does not support dynamic resources management and task partitioning.

2.2.2 AppLes Parameter Sweep Template (APST)

APST[24] is user-level middleware developed to supplement its first generation Application Level Scheduler (AppLes) [21]. The system is designed to schedule and deploy the large-scale parameter sweep applications (PSAs) on the Grid. PSAs are defined as a class of independent tasks with the same input files and a different initial state. The system provides an adaptive scheduling mechanism which could be very efficient in the dynamic Grid environment. The idea is to create a real-time Gantt chart

and schedule the tasks based on the charts. It first computes the next scheduling tasks and then creates a Gantt chart base on the computed results. For potential task executions and file transfers, it computes the estimated completion time. Then it updates its Gantt chart. The system repeats the above processes until all the computing resources have been assigned to the tasks. Finally, it schedules the tasks based on the Gantt chart. The method of updating the Gantt chart is based on the scheduling heuristics the system chooses. There are generally five types of heuristics it has implemented: self-scheduled workqueue, Min-min, Max-min, Sufferage and XSufferage.

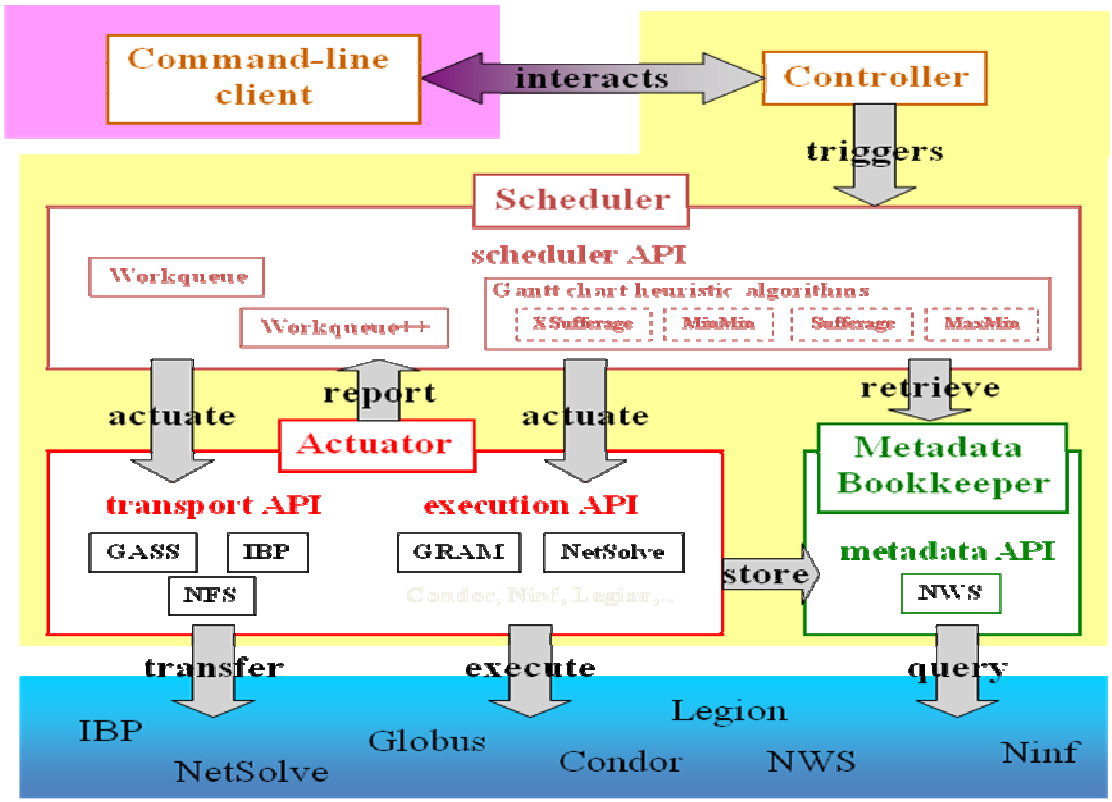

Figure 2: the infrastructure and the logical model of the APST[24], which describes different heuristics it uses

The APST provides the so-called reflective scheduling mechanism at application level, such as computing time and data transfer time by introducing the Gantt chart technique. This idea behind is for different hardware environment, different scheduling heuristics could be used. Furthermore, it cannot automatically switch from one heuristic to another once the environment changes. The users need to choose the suitable scheme manually. However it concentrates on the parameter sweep applications.

Furthermore, it does not provide the auto-partitioning function. Users have to write the specified codes to allow their tasks to be divided into sub-tasks.

2.2.3 Nimrod/G

Nimrod/G [4][19] combines its previous version Nimrod and Globus. Nimrod is used to provide user-level middleware to automatically schedule the parameter sweep applications in a single administrative domain. Globus is designed to manage the tasks across multi-administrative domains. The Globus Security Infrastructure (GSI) provides the authentication check across different domains. The Globus Resource Allocation Management (GRAM) and Dynamic-Updated Request Online Co-allocator (DUROC) can manage the scheduling problem in multi-domain and heterogeneous computing resources. The Globus Access to Secondary Storage (GASS) and GridFTP allow the user to access the data from different locations. The Globus Resource Information Service (GRIS) and Globus Index Information Service (GIIS) provide the resource registration and discovery function.

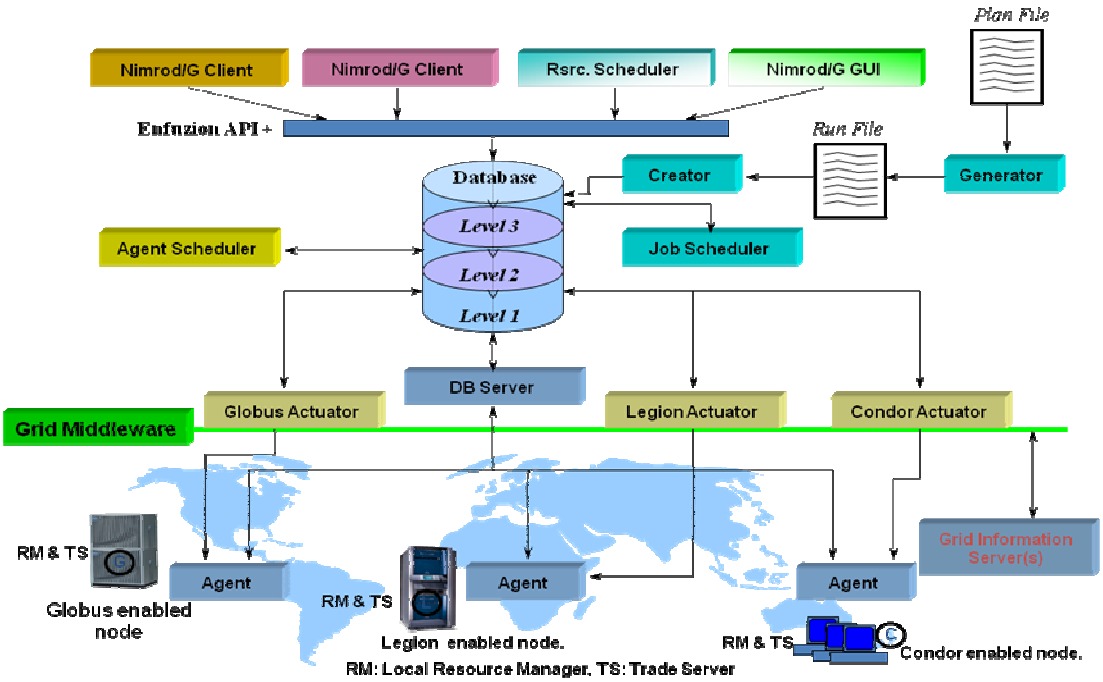

Figure 3: the infrastructure and the logical model of the Nimrod/G [19]

The scheduling algorithm is based on an economic model, which is a tree search problem associated with the cost function. It first finds a set of computing resources through the MDS [31] search and dispatches the tasks from the root to the computing resources. Then it establishes the consumption rate for each task in each machine. If the task cannot be completed before the deadline set by the user, the system will return the task and dispatch the task to a new computing resource. In an economy-based approach, scheduling decisions are made dynamically at runtime and are driven and directed by the end-users requirements. While a conventional cost model often deals with software and hardware costs for running applications, an economy model primarily charges the end user for services that they consume based on the value they derive from it. Pricing policies are based on the demand from the

users and the supply of resources is the main driver in the competitive, economic market model. Therefore, a user competes with other users and a resource owner with other resource owners.

Nimrod/G provides task partitioning support, but again it only works for the parameter sweep applications, which are independent tasks with different input parameters. It does not provide a reflective scheduling scheme and instead it uses the economic model.

2.3.4 Comparison

The related work is discussed above against the challenges stated at the beginning of this chapter. Although it is not fair to compare the other schedulers against the requirements that they were not designed for, the comparison is used to describe the design of our Reflective scheduler.

	Dynamic Management Mechanism	Task Partitioning Support
Reflective Scheduler	System Level; Automatically Switches Between Different Environment;	Data Dependent or Data Independent
Condor-G	NO	NO
Nimrod/G	NO	Only for Data Independent Parameter Sweep Applications
APST	Application Level; Provides Different Heuristics for Users to Choose	NO

TABLE 1: COMPARISON BETWEEN THE REFLECTIVE SCHEDULER WITH OTHER POPULAR SCHEDULERS

The design of the Reflective scheduler is mainly focused on two areas:

- The dynamic management
- The task partitioning support

The dynamic management here means the reflective scheduling mechanism. Due to different specifications of computing resources, different properties of task types and the different distribution of computing resources at different time, no single scheduling heuristic can provide a satisfactory performance. Even the adaptive scheduling in APST only considers the application level, which dispatches different parts of one task to computing resources by using

different scheduling heuristics. Again, it only provides different heuristics for the users to choose, not automatically.

The task partitioning support is solved in our scheduler system by providing a skeleton library, which allows users to use the existing and pre-defined parallel APIs. These APIs can help the user to hide these parallelisation details and give a robust and high standard performance. Unlike the Nimrod/G and APST, the reflective scheduler not only deals with the partitioning in Farm [14] structured programs, but in other parallel skeletons such as Pipeline [14] and Butterfly [14].

Summary

In this chapter, a group of major scheduling heuristics has been discussed as well as some widely-used schedulers. Furthermore, the thesis gives comparison for each individual heurists based on their performance. Followed by this, some popular schedulers are introduced briefly. These schedulers provide stable and good performances, however compared to our reflective scheduler two major disadvantages exist. The first does not give dynamic management in the fast-changing environment and the second does not provide efficient and convenient task partitioning support.

Chapter 3 Dynamic Management System

In the previous chapter, we discussed different basic heuristics and some popular schedulers. As we can see, these scheduling heuristics are very powerful for the static environment, even in less dynamic environments. However, in respect of computing resources and tasks, a Grid has the potential to vary continually. So there are opportunities to improve upon these traditional heuristics and schedulers for the Grid. In order to improve the performance of scheduling heuristics under these circumstances, some approaches are provided, such as a re-scheduling policy from Sakellariou and Zhao [87]. However, no matter how much updated information these schedulers use or how frequently they re-schedule, scheduling heuristics are still static. It is clear that, using a static heuristic for a dynamic environment may not be the optimal choice. We now wish to propose a dynamic scheduler for this continually changing environment. Before discussing new ideas behind the scheduler, we need to address the following two questions:

- Why ‘dynamic’?

Regarding resources or tasks, the environment of the Grid can be fast-changing. Therefore even if we know accurate information about the future of the Grid, the scheduler may not be using appropriate heuristics under these conditions. On the contrary, a dynamic scheduler chooses suitable heuristics according to different conditions. As we said before, because of the dynamism, it is almost impossible to find an optimal strategy for the whole execution period. Therefore our dynamic scheduler aims to find a relatively advantageous solution for each sub-period.

- How 'dynamic'?

  Simply speaking, the scheduler is made up of different scheduling heuristics and selects the best heuristics with its estimates of Grid load. First the scheduler must predict the next period condition of the Grid. Once the forecasting meets the pre-defined target condition, it will then switch to the relevant scheduling heuristic. So the performance of the scheduler is actually dependent on triggering conditions and heuristics integrated into the scheduler.

To achieve this goal, we will introduce an important concept in our scheduler design, reflectiveness, which gives the scheduler the ability to change its scheduling heuristics. "Reflectiveness" here means that the scheduler is able to represent its own behaviours and could be amenable to inspection and adaptation, which will provide an adaptive dynamic scheduling control for the Grid in different environments. Moreover, to support reflectiveness, we will

need the system to be able to understand what would happen in the next period so that the system can modify its heuristics in anticipation. Therefore we propose a technique which we call Blume adjustment [16] to help us forecast the condition of the Grid. In the next part of this chapter, we will explain these two concepts in details.

3.1 Implementation of Dynamism

If we consider the economic basis of a market [19], the concept of supply and demand means trading activity among people in markets. The market is a group of people consisting of consumers and suppliers for a certain service or commodity. The consumers as a group decide the demands of the service, and suppliers, as a group determinate the supply side. Users' tasks are similar to consumers and computing resources are similar to suppliers in terms of markets. Also the task dispatching process can be regarded as a transaction between the consumer and supplier. Because of these characteristics, the Grid can be regarded as a market. The construction of an efficient Grid is similar to constructing an efficient market.

Some market-based heuristics, such as Nimrod/G[19], use the "price" to control the supply and demand in the Grid market. However, these heuristics miss some important points:

- It is very difficult to define the concept of the price in the Grid environment.

- By using the price concept, it is very difficult to guarantee that a Grid will have a good load balance and users will be happy with the quality of the service.
- Most importantly, the Grid is not an efficient market. In the traditional market, if the price increases, the demand will decrease due to the customers' budgets. Meanwhile, if the price decreases, the supply will decrease because the number of suppliers will reduce. But in the Grid, supplies will not change much due to the changes in the prices, as the price does not affect whether computing resources provider offers services or not, as well as the demands. As the supply and demand are inelastic, it is not easy to construct an efficient market by purely using the price.

Due to above reasons, the price itself is not capable of keeping the Grid efficient. In contrast, our new scheduler is able to change allocation heuristics according to the current relationship between demand and supply, which we think is more feasible in current conditions. Therefore two key issues are worth noting: how do we change the allocation heuristics and which heuristics should be used. The reflectiveness and the welfare economics are used to construct a more practical and efficient Grid market.

3.1.1 Reflectiveness

The idea of reflectiveness comes from dynamism, which is now widely used in programming design. In [46], reflectiveness is a property that "*can be used for self-optimization or self-modification*". The reflective system can modify or

optimize its own conditions according to the state of the process during execution. Similarly, the scheduling heuristic driven by reflection can be called the reflective scheduler.

Reflectiveness can also be used to provide different solutions given the dynamically changing nature of the system. For the reflective scheduler, the scheduler uses some scheduling heuristics, *X*, to dispatch tasks. If circumstances change and the scheduler needs to adopt different heuristics, *Y*, which has different scheduling names, using the concept of reflection, the scheduler can determine when to switch schedule heuristics in the scheduler. Moreover, the scheduler could be designed to provide information regarding which method is being used for what purpose. The scheduler, depending on what it has to do, will select the required scheduling heuristic and switch to it. The system is able to change its own scheduling heuristics accordingly using the reflectiveness property.

3.1.2 Heuristic Design

The second point we need to consider is how to find the optimal solution in the Grid environment. The scheduler needs to balance the following two criteria:

- The users' requirements are satisfied
- The computing resources are fully utilized.

So in the reflective scheduler, we design, in general, two kinds of scheduling schemes: application-oriented and system-oriented. Application-oriented

means the scheduling scheme is optimized for the best *response time*, which is used to measure the waiting time for a task to execute. The alternative approach is termed system-oriented. Here the heuristic is designed for the best *balance level*, which is used to measure the load balance of computing resources. The scheduler will choose between those two different heuristics in different conditions. If the computing resources are much decreased compared to task queue and reach the triggering level, the scheduler turns to the system-oriented heuristic which will increase the *balance level*. If the computing resources are much increased and reach the triggering level, the scheduler turns to the application-oriented heuristic which will decrease the *response time*. In this case, the Grid can be always set to a condition, which maximises in the total efficiency.

3.1.3 Optimisation of design

The combination that allocates a set of resources, such as goods, or assets to a set of individuals, can be massive. Here we consider applying the Welfare Economics to scheduler design. Optimisation in a multi-objective frame work such as this one is a complex subject. Key work in this area was introduced by Pareto [84]. A Pareto improvement means from one allocation to another will improve at least one individual without impairing other individuals. The allocation is called Pareto efficient or Pareto optimal when no further Pareto improvement is available. The Pareto efficiency and Pareto optimality [84] originated from economics research plays a key role in modern engineering,

game theory and other social sciences. However Pareto efficient is too strict to implement, as no single individual's benefit will be impaired.

An alternative criteria is Kaldor-Hicks efficiency [84] which will be used in this design. Using Kaldor-Hicks efficiency, "*an outcome is more efficient if those that are made better off could in theory compensate those that are made worse off and lead to a Pareto optimal outcome*."[84] Thus, a more efficient outcome can, in fact, leave some people worse off. But in a long term view, those individuals impaired will be compensated at some future time and some individuals currently advantaged will be damaged in the future. So generally, these effects are offset in the long run and eventually everyone benefits.

In the Grid, each computing resource is encapsulated into one node so that one node can provide one service at a time. Scheduling is the mapping of tasks to nodes. Therefore we define the Grid environment as follows:

$$Task = (t_1, t_2, \ldots, t_n)$$

$$\mathrm{Re}\,sources = (r_1, r_2, \ldots, r_m)\, nm$$

$$Queue = (q_1, q_2, \ldots q_m)$$

$$Completion = (c_1, c_2, \ldots, c_n)$$

Where $Task$ stands for the n n tasks from n users, $\mathrm{Re}\,sources$ stands for the m available computing resources, $Queue$ stands for the maximum queue can

be hold in one computing resources and $Completion$ stands for the response time for each task.

Definition 1: The matrix $Allocation = \begin{pmatrix} tr_{11} & \cdots & tr_{1m} \\ \vdots & \ddots & \vdots \\ tr_{n1} & \cdots & tr_{nm} \end{pmatrix}$ defines one possible allocation, so the tr_{ij} means the task t requests the resource r. So to any task $i = 1, 2, \ldots, n$, the matrix needs to satisfy:

$$0 \le tr_{ij} \le q_j \text{ and } 0 \le \sum_{i=1}^{n} tr_{ij} \le q_j$$

This definition means if one allocation is potentially available, the tasks cannot access the number of the available resources.

Definition 2: If there is one completion time $Completion^* = \left(c_1^*, c_2^*, \ldots, c_n^*\right) \in Completion$ and one allocation matrix $Allocation^* = \left(tr_1^*, tr_2^*, \ldots, tr_m^*\right) \in Allocation$ can satisfy:

$$U = \frac{1}{\min\limits_{Allocation^* \in Allocation} D(tr_i)}$$

$$\text{And } B = \min_{Allocation^* \in Allocation} Completion^* \left(\sum_{j=1}^{m} tr_{ij}\right)$$

U stands for the reverse of the variance of queues for each resource, which reflects the balance level of the Grid. The larger this value, the more balanced

the system is. U can be regarded as the goal of the system-oriented heuristic. B represents the total response time of each task. Again, the smaller this value is, the faster the system can execute tasks. Again B can be regarded as the objective of the application-oriented heuristic.

3.1.3.1 Demand

We start our research of the Grid from the consideration of the consumer. In terms of the Grid, the demand is the amount of computing resources that tasks need. The factors which will affect the decision are:

- Balance Level: the higher the Balance Level is, the better the Grid's performance will be. When the other factors are unchanged, the more tasks come in, the less computing resources are available. Therefore, the Balance Level will decrease accordingly.
- Response Time: the lower the response time, the better the Grid's performance. When other factors are unchanged, the more tasks that come in, the longer they need to wait in the queue. Therefore, the response time will increase accordingly.

As the efficiency level is in direct proportion to the Balance Level and inverse ratio of the response time, the demand curve for the Grid can be displayed as: the less the demand for service, the higher the efficiency level will be, and vice versa.

3.1.3.2 Supply

Here we need to find out the factor which decides the amount of the supply. It's worth noting that the supply is the amount of the service the supplier willing to provide. In terms of the Grid, the supply is the amount of computing resources provided. The factors that will affect the decision are:

- The Balance Level: the higher the Balance Level is, the better the Grid's performance. When the other factors are unchanged, the more available computing resources come in, the more supplies are available. Therefore, the Balance Level will increase accordingly.

- The Response Time: the less the response time is, the better the Grid's performance. When the other factors are unchanged, the more available computing resources come in, the shorter the queue is. Therefore, the response time will decrease accordingly.

Again, as the efficiency level is in direct proportion to the Balance Level and inverse ratio of the response time, so the demand curve for the Grid can be displayed as: the more computing resources provided, the higher the Efficiency Level is, and vice versa.

3.1.3.3 Efficiency of the Market

According to this theory, the system-oriented heuristic and the application-oriented heuristic are competing to construct an optimal condition under the above constraints. In other words, these two heuristics are trying to improve

the total efficiency of the system. In some situations, the Grid system can achieve this purpose through the system-oriented heuristic. Under these circumstances, computing resources can get better load balance but users may need to wait for a longer execution time. In the other situations, the Grid system can reach the goal by using application-oriented heuristic. In this case, users can get shorter response time but computing resources may find a worse load balance. However in a long term view, these effects can be offset and the Grid system can achieve a better performance without impairing the benefits of both users and computing resources.

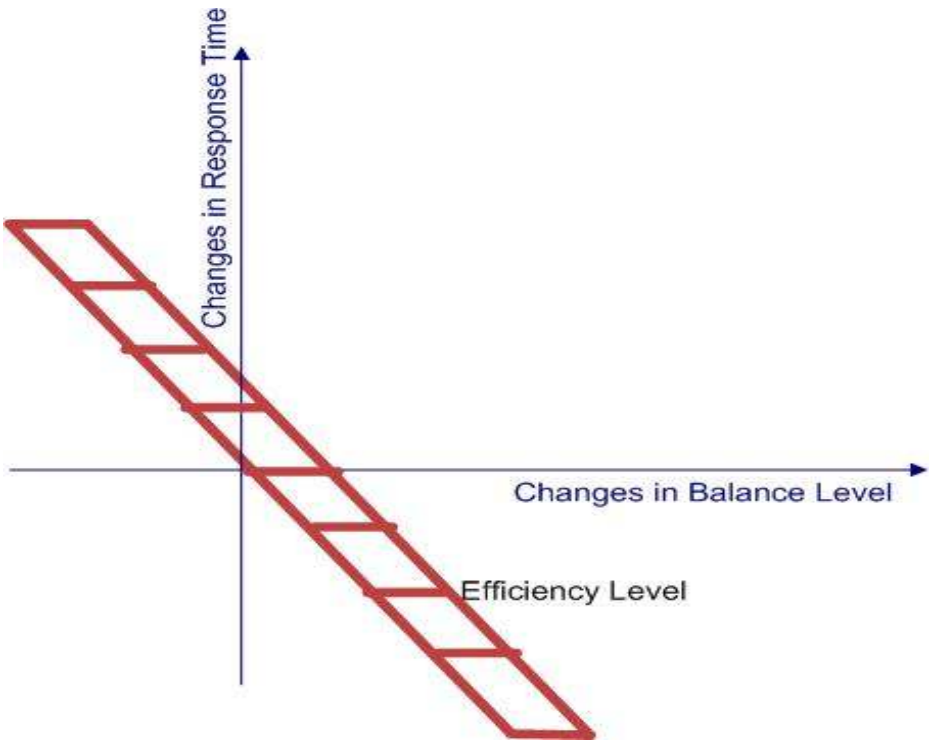

Figure 4: The Efficiency Level of the grid changes due to the changes of both the demand and the supply, and in a long term view, the system can achieve a better performance without impairing the benefits of both users and computing resources

3.2 Forecasting the Grid Situation

From the above section, we learned that the system can change its strategies according to the environment. Thus it requires the system to have the ability to predict the situation for the next period. Performance forecasting is the basis for the scheduler to provide high standard and efficient services. If the system knows that in the next period a huge number of tasks will come in and a huge number of computing resources will leave the Grid, the system then needs to switch to the system-oriented strategy in order to fully utilize all available resources.

Strictly speaking, because of evolution of Grid, the previous situation is very difficult to replicate, so the simulation of the past environment is almost impossible. Furthermore, it will be more difficult to precisely forecast the future condition of the Grid. However, we are able to solve the problem by using statistical methods. As we know, statistics try to find some inner characteristics of events by using large samples of the historical data.

Using historical data to forecast the future condition can be unreliable. But this kind of forecasting can help to solve problems, such as weather prediction. The more historic data available, the better researchers can understand factors affecting the weather and the more precise the results which can be achieved.

However, it is worth noting that forecasting is only useful for an event following certain rules. If an event does not follow certain rules, like the exchange rate,

it will be useless to try to forecast by using historical data. No matter how much historical data has been obtained, it is impossible to forecast what happens next.

All entities in the Grid, such as tasks and computing resources, do not look like a stochastic system. In fact, they follow certain rules. Activities which happen in the Grid change periodically. For example, in a certain location, the activities for the Grid are more frequent in day time than at night. So it is possible to record all the statistic data of Grid activities, and then predict the future situation.

3.2.1 Blume Regression

This idea [16] derives from the market risk prediction for economic usage. In the financial market, people think the market risk should follow a mean reversion rule. This implies that the value of the market risk has a certain tendency. To exanimate the question of time-invariant beta, we run inter-period regressions on the betas. Here the beta is a quantitative measure of the volatility of a given stock, mutual fund, or portfolio, relative to the overall market. A beta above 1 is more volatile than the overall market, while a beta below 1 is less volatile.

In the computing resource forecasting, the situation is very similar. Each computing resource can be regarded as individual stock in the market, while the ratio of available computing resources can be regarded as market risk.

It is worth noting that Blume regression is one approach to forecasting the market risk and there are plenty of other techniques can be used in the future, such as Bayesian regression [16], moving average models [23], ARCH model [93] and Garch models [95].

First, we can collect the pair $[HR_t, HR_{t-1}]$ which stands for the ratio of available computing resources at time t and the ratio of available computing resources at previous time $t-1$. After this, we carried out the regression on this pair

$$HR_t = \alpha + \beta HR_{t-1} + \varepsilon$$

Where α and β are least square regression coefficients and ε is a random disturbance term. HR_t and HR_{t-1} stand for the ratio of available computing resources in the current and previous periods.

If in the above equation, the absolute value of slope coefficients is less than one, the collecting series is stationary. Then the equation of simple linear regression accordingly run for a sub-period is:

$$HR_t = \alpha + \beta HR_{t-1} + \varepsilon \text{ for } t = 1, 2, 3, \ldots n$$

Using the HR_{t-2} instead of HR_{t-1} recursively, we can get the

$$HR_t = \alpha + \beta HR_{t-1} + \varepsilon$$

$$HR_{t-1} = \alpha + \beta HR_{t-2} + \varepsilon$$

$$HR_t = \alpha + \beta(\alpha + \beta HR_{t-2} + \varepsilon) + \varepsilon$$

$$HR_t = \alpha + \beta(\alpha + \beta(\alpha + \beta(\alpha + \beta HR_{t-2} + \varepsilon) + \varepsilon) + \varepsilon) + \varepsilon$$

….

Then the expected value of HR_t can be derived from HR_0, the initial ratio, where

$$E(HR_t) = \alpha + \alpha\beta + \alpha\beta^2 + \alpha\beta^3 + \alpha\beta^4 + \alpha\beta^5 + ... + \alpha\beta^{t-1} + \beta^t E(HR_0)$$

$$E(HR_t) = \alpha(\frac{1-\beta^t}{1-\beta}) + \beta^t E(HR_0)$$

$$\lim_{t\to\infty} E(HR_t) = \lim_{t\to\infty}(\alpha(\frac{1-\beta^t}{1-\beta}) + \beta^t E(HR_t))$$

$$= (\frac{\alpha}{1-\beta})$$

For the second step, after discovering the stationary of the time series, we headed to the random walk test by using a t-statistical test. In the previous test, the condition $-1 < \beta < 1$ was crucial for stationary. If β is equal to 1 and α is equal to 0, the original series becomes

$$HR_t = HR_{t-1} + \varepsilon$$

This is an example of a non-stationary process known as random walk. Therefore, we tested the null hypothesis where α is equal to 0 and β is equal to 1. For the tendency test, the regression results demonstrate that the intercept coefficients are significantly different from zero. The slope

coefficients are also significantly different from unity. These findings suggest a definite regression, which means the time series of computing resources does not follow a random walk.

3.2.2 Testing Result

The thesis has tested the methodology based on one financial company. The company has 59 personal computers and two servers. The test first investigates the percentage of computers available during different periods of a day. “Available” means here that if the CPU usage is below 80%, the computer will join the Grid environment. Then the test examines the performance of the Blume adjustment.

In our tests, we divided the day into four periods: 0:00 to 6:00, 6:00 to 12:00, 12:00 to 18:00 and 18:00 to 24:00. Then we observed the number of computing resources in each period for four working days. Finally we regress the available ratio of computing resources at 0:00 to 6:00 of the first day on second day’s data, and so on. The intercept coefficient and slope coefficient standard deviations are reported in parentheses respectively. We can get

- **0:00 to 6:00**

$$HR_1 = 0.87693 + 0.37918 \times \mathrm{HR}_0 + \varepsilon$$

(0.096547) (0.083407)

$$HR_2 = 0.92918 + 0.154444 \times \mathrm{HR}_1 + \varepsilon$$

(0.077706)(0.052976)

$$HR_3 = 0.88869 + 0.21577 \times \mathrm{HR}_2 + \varepsilon$$

(0.090675)(0.061064)

With average slope coefficient is 0.24979 and average intercept coefficient is 0.89494

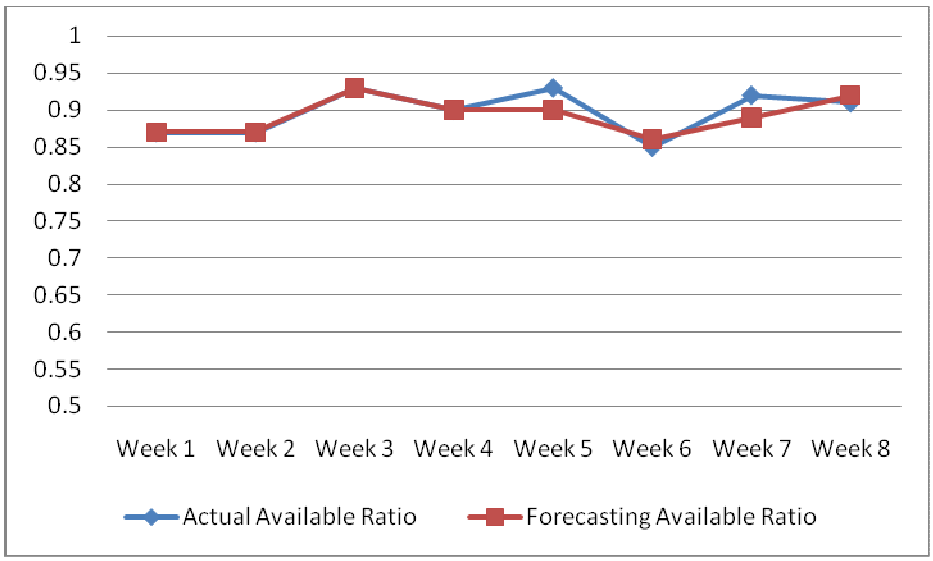

Figure 5: Comparison between the actual available ratio of the computing resources and the forecasting ratio during 0:00 to 6:00

- **6:00 to 12:00**

$$HR_1 = 0.47459 + 0.29809 \times \mathrm{HR}_0 + \varepsilon$$

(0.074023)(0.065566)

$$HR_2 = 0.67338 + 0.1846 \times \mathrm{HR}_1 + \varepsilon$$

(0.07817)(0.082401)

$$HR_3 = 0.67151 + 0.15869 \times \mathrm{HR}_2 + \varepsilon$$

(0.031393)(0.029745)

With average slope coefficient is 0.37672 and average intercept coefficient is 0.59357

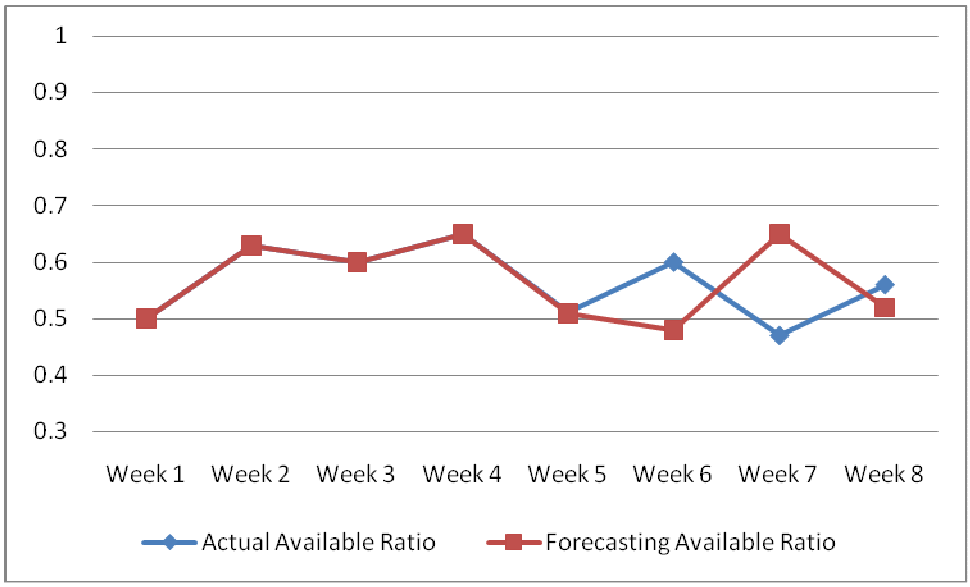

Figure 6: Comparison between the actual available ratio of the computing resources and the forecasting ratio during 6:00 to 12:00

- **12:00 to 18:00**

$$HR_1 = 0.24284 + 0.05509 \times \mathrm{HR}_0 + \varepsilon$$

(0.073386)(0.039335)

$$HR_2 = 0.60399 - 0.0361 \times \mathrm{HR}_1 + \varepsilon$$

(0.062981)(0.037123)

$$HR_3 = 0.43573 + 0.16844 \times \mathrm{HR}_2 + \varepsilon$$

(0.038215)(0.02288)

With average slope coefficient is 0.06248 and average intercept coefficient is 0.42752

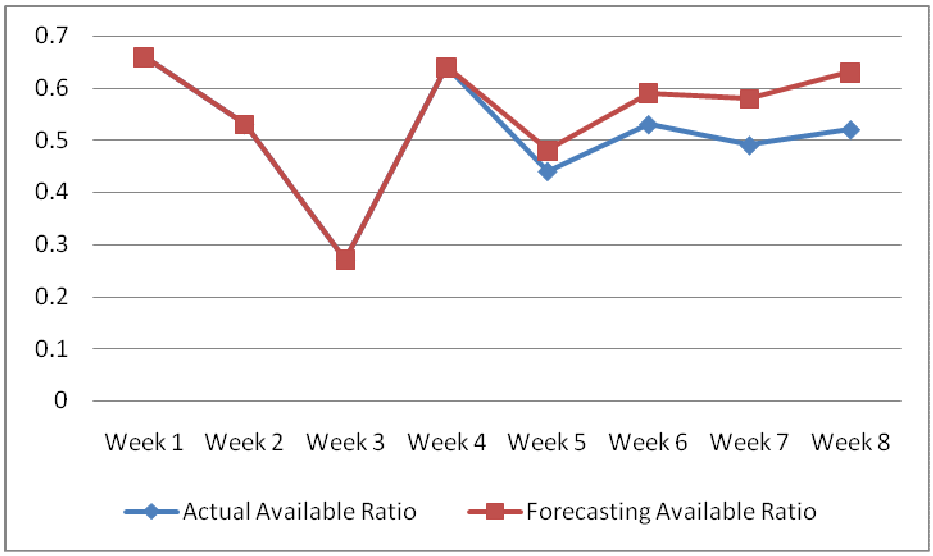

Figure 7: Comparison between the actual available ratio of the computing resources and the forecasting ratio during 12:00 to 18:00

- **18:00 to 24:00**

$$HR_1 = 0.7711 + 0.11407 \times \mathrm{HR}_0 + \varepsilon$$

(0.138343)(0.114604)

$$HR_2 = 0.72397 + 0.18205 \times \mathrm{HR}_1 + \varepsilon$$

(0.078818)(0.063552)

$$HR_3 = 0.7025 + 0.13731 \times \mathrm{HR}_2 + \varepsilon$$

(0.065925)(0.062487)

With average slope coefficient is 0.14447 and average intercept coefficient is 0.73252

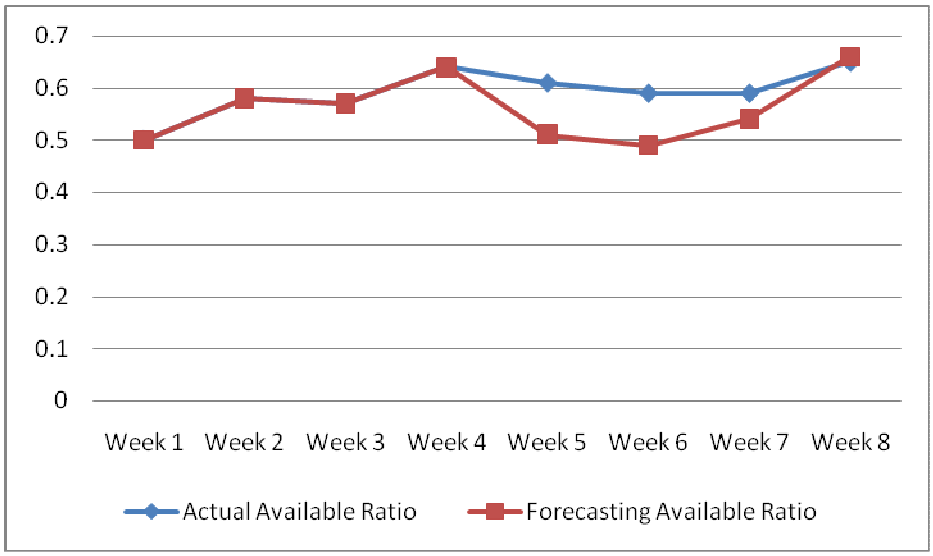

Figure 8: Comparison between the actual available ratio of the computing resources and the forecasting ratio during 18:00 to 24:00

In this test, in the best case, for 0:00 to 6:00, the error in forecasting is less than 8% and in the worst case, the error in forecasting is less than 15%. This test shows that there is indeed some tendency existing in the ratio of the available resources. Using this information, the system can roughly understand the condition for the next period. Due to the sample size of the historical information obtained, the performance can be improved in the future

once the system has stored enough data. Also with better regression techniques, the forecasting power can be further improved.

Summary

In this chapter, we introduce the central idea, reflection which will be incorporated into our reflective scheduler. The reflectiveness is applied to the selection of scheduling heuristics. It gives the scheduler the ability to dispatch the task in a dynamic environment. The idea of reflective can be used to deal with the dynamism of the Grid environment. Finally, to support the reflective heuristic, the scheduler needs to estimate the condition of the Grid in the next period. We have used Blume adjustment to forecast the condition of the environment.

Chapter 4 Mathematical Simulations

We start this chapter by considering the range of methods available for the measurement of scheduler performance. Basically there are three frequently used approaches [111] in computer system evaluation, which are measurement methods, analytic methods and simulation. Briefly, what the measurement approach does is mainly to observe existing systems, collect statistical data during the run time and use these results to evaluate performance of the system. The observation data are used to plot the system's parameters to satisfy specific criteria. Analytic methods describe the real system in terms of models using mathematical language, and then solve the mathematical formula to find its optimal parameters. For systems such as networks, queuing theory is one of the most popular mathematical methods used to describe the network related systems, such as computational Grids. The reason for this is that queuing theory mainly focuses on problems such as the waiting queue, serving queue and so on. It is clear that in the Grid, the problems such as submitting jobs, waiting for the results and requesting I/O devices are all queuing problems. In this case, it is an ideal mathematical tool

for Grid performance analysis. However, in some cases, conditions of the Grid are too complicated to be dealt with. In this case, simulation becomes the only feasible method to analyse the systems. Unlike analytic methods, this approach will not generate any bias based on the assumptions. Although it takes longer, it will generate a more accurate model. So in this chapter, we first analyse the system's performance through queuing theory and then use simulations to verify the results in the next chapter.

4.1 Evaluation Model

Grid Technology has rapidly developed, and is widely used in different industries. Obviously the Grid systems become more and more complicated, which leads to many challenges in timing and space constraints. We have introduced Petri Net [74][76] technology in order to study the performance of a Grid, to simplify the construction, execution, management and monitoring of a Grid, and to strengthen the resource sharing and cooperation. A Petri net is a graphical and mathematical modelling tool. Petri nets are promising tools for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a

mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. It is able to support the workflow dynamic modelling and dynamic scheduling. Because the Grid environment is dynamic, workflow dynamic modelling and scheduling thus plays a key role in the investigation of the utility of particular scheduling methods in the Grid.

4.1.1. Single Server Model

A Grid is made up of a number of single nodes, and each node can be regarded as an independent server. For a specific server, its processing ability or performance is fixed. This characteristic is reflected in two ways:

1. It is known in advance whether the server can process a particular job

2. It is also known in advance how fast the server can process the job.

The single server model is used to evaluate this kind of problem. It can be shown by the following figures:

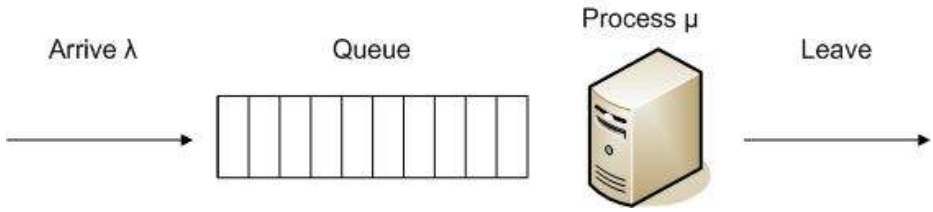

Figure 9: the single server model described by queuing theory

There is only one server in this model, so all the requests needing to be processed by the server can be put into a single queue. The process ability for the server is μ and the distribution of the request arrival is λ. In this case, we need to answer the following questions:

- the average length of the queue in the system
- the average waiting time for a request in the queue
- the availability of the system

This model is very useful to evaluate a single server's performance. When we consider a single node in the Grid, we can use this model. The request comes from the remote scheduler to this node and this request is to execute one job submitted by one user. The requests in a queue wait for the server to execute. It is worth noting that different requests in the queue need different execution times by the server. So to describe the system correctly, we need to add one more parameter, the execution time for the request. Then the model is changed to:

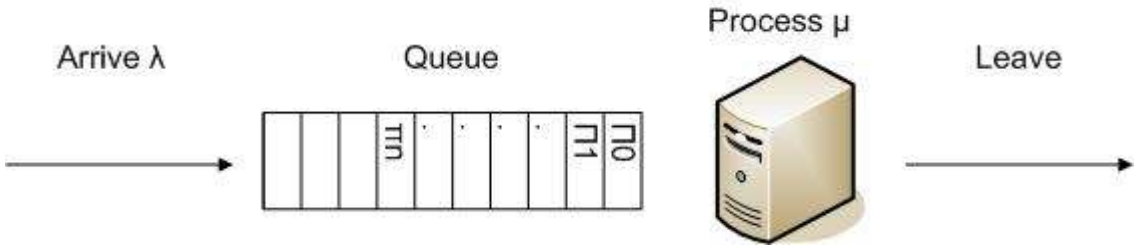

Figure 10: the single server model with different queuing time described by queuing theory

Suppose the scheduling scheme is First Come First Served and only one request can be processed at one time, which is that the request will be executed after the previous one is finished. The waiting time for the last request entering the queue is:

$$\pi = \sum_{i=1}^{n} \pi_i$$

The single server model can not only describe the performance of the single node in the Grid, but also can evaluate the performance of the Grid. From the view of the users of the Grid, it is a well integrated system. After the users submit their requests to the Grid, it processes them and returns the results to the users. So the Grid can be abstracted as a single super node, which is made of a number of nodes.

4.1.2 Queued Network Model

A more general model of a computational Grid includes requests needing multi-resource formats in a queued network. It is normal that in many conditions there is a queued network. For example, one computational request, firstly needs to be calculated in the high performance node, and then translates the result into figures following storage of the final result. All in all, this kind of phenomenon frequently exists in the Grid.

A queued network is a directed queue. In terms of a queuing network, requests enter the network and leave after they are finished. Different requests need a different number of nodes, from two or three to thousands.

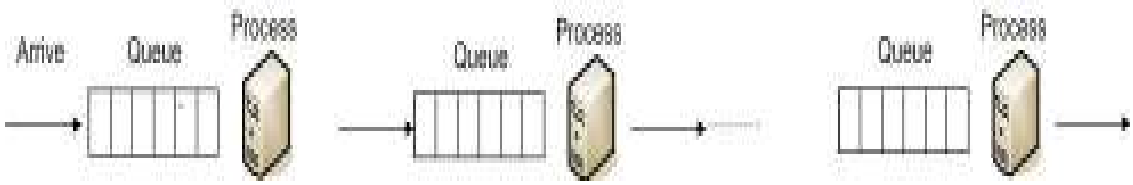

Figure 11: the basic model of the queuing networking system described by the queue theory

To regard the Grid as a single node will simplify the problem. In fact, the Grid contains a large number of resources. When the user submits a request to the scheduler, it will hand it on to a lower level system to deal with. So the user using the same scheduler formats a queued network and all the requests submitted to this scheduler need to be put in a queue. The scheduler can divide the requests into different queues based on the request's requirements.

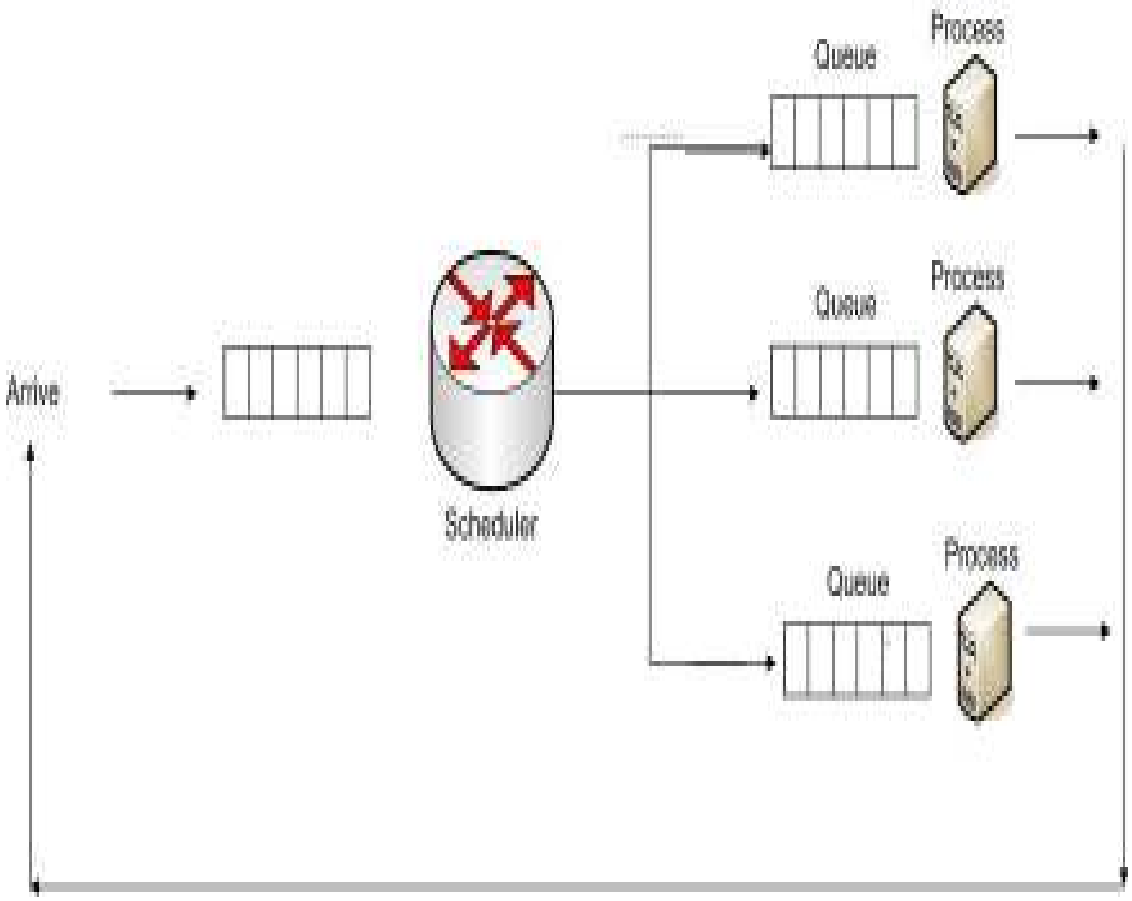

Figure 12: the basic model of the Grid system described by the queuing theory

If the request can be finished in one node, the request will not be returned to the queue again. Otherwise, if the request needs p nodes, the execution time becomes:

$$\pi = \sum_{i=1}^{p} \pi_i$$

Where π_i is the time consumed by one single node, including the waiting time and processing time. The waiting time does not just mean the waiting time in the queue but also the time spent in matching and scheduling.

4.2 System Model for the Scheduler

First of all, in order to describe the whole Grid system, we have constructed a system model with the minimum numbers of function units: Tasks from the users, RS (Reflective Scheduler) and CR (Computing Resources). In this system model, one RS is used to execute the scheduling heuristics. It receives all the incoming requests, classifies those requests, and allocates them into n different queues. I this case, we have n equal to two: High Priority Queue and Low Priority Queue. If a task is not successfully completed, the system needs to reschedule them and puts them in the high priority queue. Then the priority queue is put into the scheduling heuristic, and finally realizes a reflective task allocation. The whole process is as follows:

- The user submits a task to the RS
- The RS fetches the information from the task and allocates the task to different priority queues.

- The RS judges the current situation of the queue and CRs to decide which scheduling heuristics to use, Application-Oriented or System-Oriented, and then
- The RS schedules the request to a specified CR.

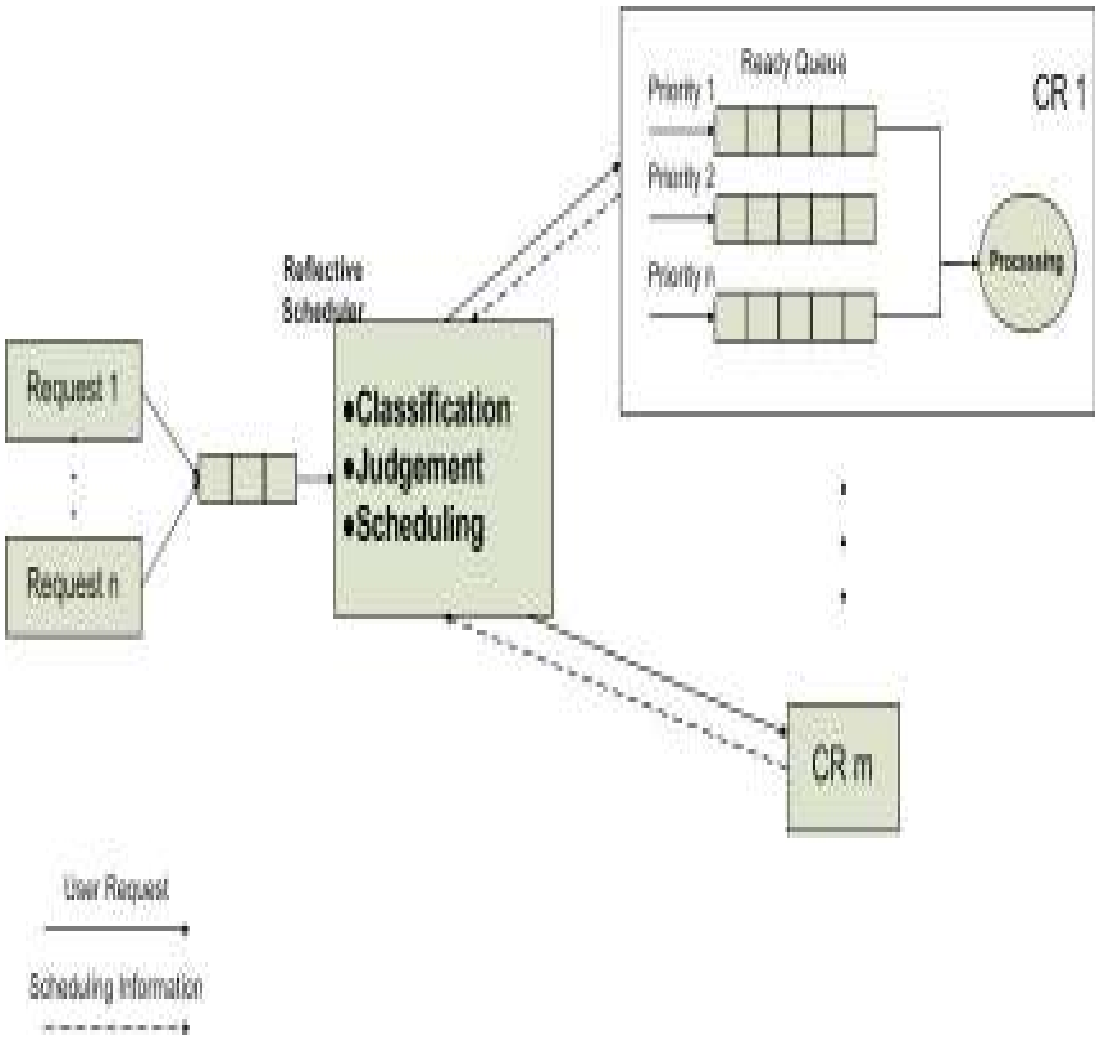

Figure 13: Simplified System Model for Grid using the Petri Net

4.3 Stochastic High Level Petri Net Model

We will introduce the SHLPN (Stochastic High Level Petri Net)[11] modelling method and analysis technology to evaluate the performance of the Grid system based on the reflective scheduling method. We have provided a system model based on reflective scheduling and now the following are the specifications of this model:

1. Arriving requests are classified and allocated into two different queues according to the priority. For $1 \le i < k \le n$, priority i is higher than priority k. Request with priority i is denoted by r_i.
2. The Grid system contains m CRs and V_j represents the j_{th} CR.
3. Each CR has two ready queues with different priorities and both queues have a maximum queue length l.
4. The task arrival rate obeys a Poisson process [74][77][112]. Let us assume that the arrival of the request r_i form a Poisson stream with parameter λ_i, in other words, the arrival intervals of request λ_i obey an exponential distribution with parameter $\frac{1}{\lambda_i}$. When the queue reaches its maximum queue length, the CR refuses to accept any requests and the tasks need to be sent back to the high priority queue to re-schedule.

5. Task's processing times with different priorities in different CRs again obey an exponential distribution [74][77][112] but with different parameters. The average processing time for request r_i in CR V_j is μ_{ij}.

In this model, the rectangles denote timed transitions and the black bars denote immediate transitions. Tasks arrival and service are represented by timed transitions associated with the exponential distributed firing time. Tasks being scheduled to the CRs and competing for computing resource are represented by immediate transitions with zero firing time. The circles in the model denote places, and they contain tokens denoted by black dots. The ready queues are represented by places, and the numbers of pending processes in the queues are represented by the marking of those places. Tokens in this SHLPN model can represent either CRs or computing resource, and computing of different priority levels can be represented by tokens of different colors. In this model, the first subscript of the symbols represents the category of requests, while the second subscript of the symbol represents the destination CR. If there is only one subscript, it only indicates the category of requests.

The meanings of the transitions and places in Figure 13 are described in the following ($1 \le i \le n, 1 \le j \le m$):

• Transitions:

c_i: models the arrival of requests r_i at rate λ_i.

d_{ij}: models scheduling requests r_i to CR V_j. It can be associated with firing probabilities since there may be multiple destination CRs available for an incoming request.

h_{ij}: models selecting the processes of priority i to execute by CR V_j.

s_{ij}: models processing requests r_i by CR V_j. Its mean service rate is at μ_{ij}.

dc_j: models the matching value of the CR V_j

• Places:

f_i: models the partial function of the request scheduler to distribute request r_i. The aggregation of $f_i (1 \le i \le n)$ models the whole content-aware dispatcher.

q_{ij}: models the ready queue of priority i in CR V_j with capacity b_{ij}.

w_{ij}: models the running state of a process of priority i at CR V_j.

v_j: models the processor resource of CR s_j. The tokens in v_j represents the multi-host instance.

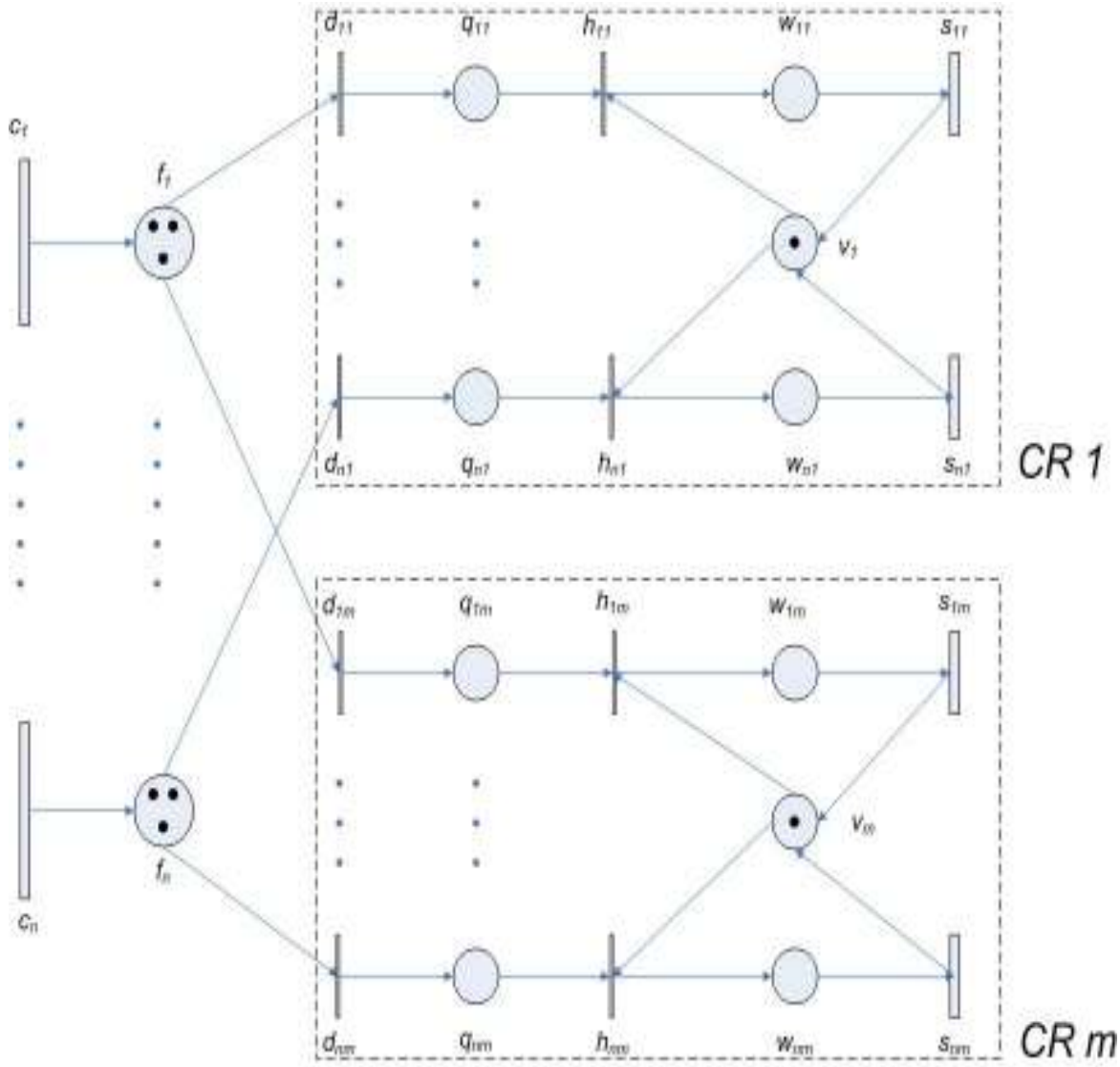

Figure 14: SHLPN Model for the Grid System

4.4 Model refinement

To analyze the performance of the SHLPN model [54] in Figure 14, we construct the corresponding MC (Markov Chain)[76] of the model. Based on the MC and state transition rates, we can construct the state transition matrix

and obtain all the steady-state probabilities to calculate the performance measurements. The software package Stochastic Petri Net Package (SPNP) [28] was developed to simulate this type of system based on this idea. This package can be used directly for the performance analysis of the SHLPN model when the size of the model is not too large. However, the SHLPN model with n places of queues is generally equivalent to an n-dimension MC. So the MC of the model in Figure 14 has $m \times n$ dimensions. The state space of the MC grows exponentially with the increase of m, n and b_{ij}. The state space is so large that solving the state equations is impossible for practical computing systems. In this report, we use SPNP to analyze the SHLPN model and explore the performance benefits of reflective scheduling policy.

In order to reduce the complexity of the model solution, we simplify the structure of the SHLPN model in Figure 14 using transition enabling predicates and rate functions. In this way, a complicated model can be equivalently transformed into a compact model. Figure 15 shows the model refined from Figure 14

In Figure 15 we deleted places w_{ij}, v_j, transitions h_{ij}, and some relevant arcs and tokens ($1 \le i \le n, 1 \le j \le m$), compared with Figure 14. The competition relations among the priority queues that originally expressed those discarded net elements are directly described in the enabling predicates and firing probabilities of transition s_{ij}. Consequently, in Figure 15, whether transition

s_{ij} is enabled or not, not only depends on the marking of the place q_{ij}, but also depends on the markings of other places in the same CR.

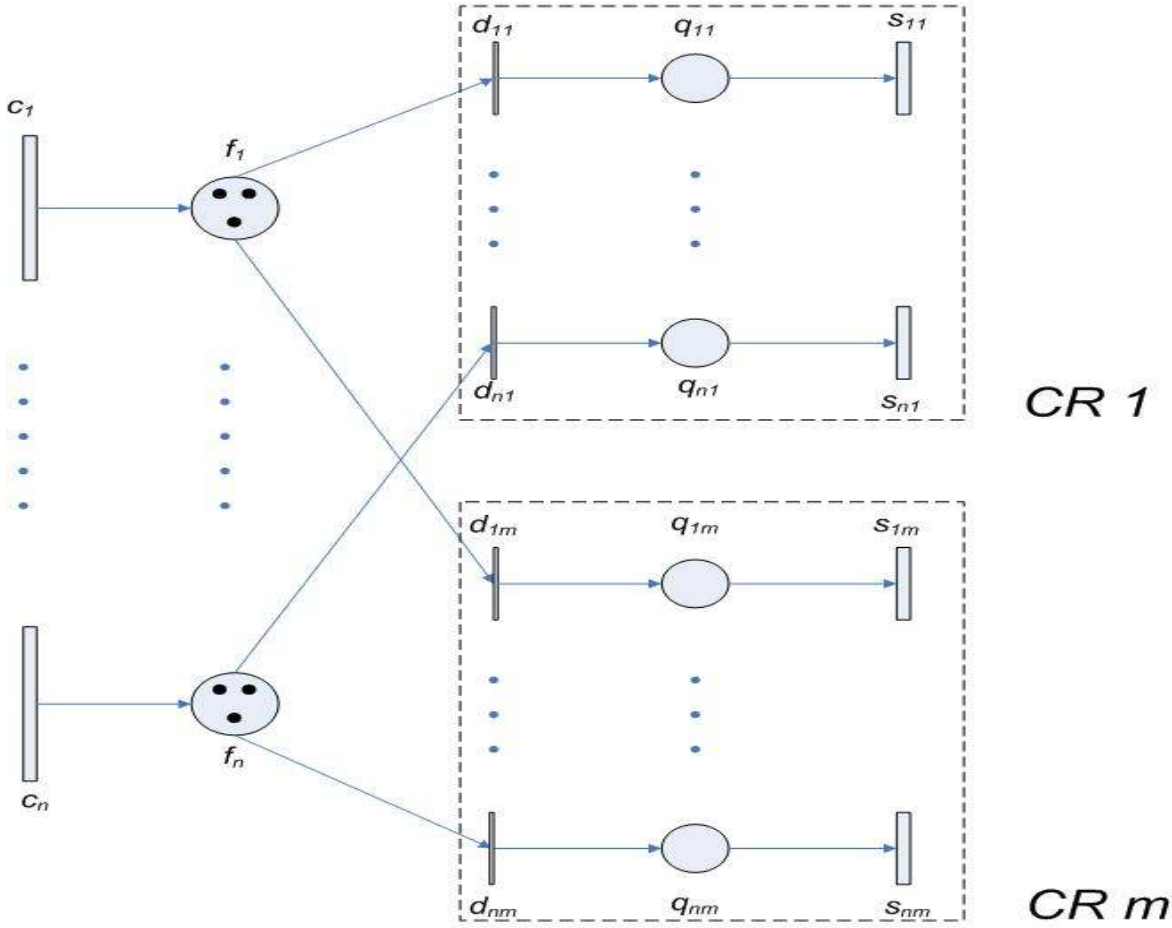

Figure 15: Compact SHLPN Model for Grid System which is simplified in order to avoid space exploration problems.

4.5 Reflective scheduling policy and performance metrics

There are various scheduling policies available for the scheduler when applied to Grid systems. In this section, we first introduce the CR process policy and

two scheduling policies concerned in this report, and then propose the reflective scheduler based on them. Metrics considered in the performance analysis are also formulated.

4.5.1 Policy description

4.5.1.1 CR requests scheduling policy

We consider the strict priority policy for the CR processes at the back-end in the Grid system. In Section 5.1.3, the priority levels used in this policy will be considered. The following notations are used in this chapter: $M(x)$ is the marking of the place x, i.e., the number of tokens contained in x under the marking M; M_0 is the initial marking of the model.

• Strict priority (SP) policy

This policy schedules all higher-priority requests before lower-priority requests even when low-priority ready requests are waiting. The strategy of SP scheduling can be described in the enabling predicate and firing probability of transition s_{ij} in the SHLPN model of Figure 15.

The enabling predicate of s_{ij} is

$$(M(q_{ij}) > 0) \vee (\forall k, 1 \le k < i \le n, M(q_{kj}) = 0)$$

Specifically, the enabling predicate of s_{1j} is simply $M(q_{1j}) > 0$.

The firing probability of s_{ij} is

$$p(s_{ij}) = \begin{matrix} 1, if \ \ i \in Q; \\ 0, \ otherwise; \end{matrix}$$

where $Q = \left\{ k \mid M(q_{kj}) > 0 and M(q_{lj}) = 0, \forall l, 1 \le l \le k \le n \right\}$

4.5.1.2 System-Oriented and Application-Oriented policies

We consider the typical policies System-Oriented and Application-Oriented for scheduling.

• System-Oriented policy

In this policy, the CR with the lowest workload (loaded with fewest user tasks) is selected as the scheduled destination. The scheduler does not consider the priority levels of requests, and request classification and prioritization are only performed at the CRs. The System-Oriented strategy can be expressed by the enabling predicate and firing probability of transition d_{ij} in Figure 15. Note that the category and priority of the requests are not explicitly known before they enter the CRs. The aggregate of the ready queues in CR V_j is denoted by q_j with capacity b_j ,and $b_j = \sum_{j=1}^{n} b_{ij}$, $M(q_j) = \sum_{k=1}^{n} M(q_{kj})$.

The enabling predicate of d_{ij} is

$$(M(q_j) < b_j) \wedge (\forall k, 1 \le m, (M(q_j) \le M(q_k)) \vee (M(q_k) = b_k))$$

The firing probability of d_{ij} is

$$p(d_{ij}) = \begin{cases} 1/|Q|, if \quad j \in Q; \\ 0, otherwise; \end{cases}$$

where $Q = \{k | M_i(q_k) = \min(M_i(q_l), M_i(q_2), \ldots, M_i(q_m) and M_i(q_{ik}) < b_{ik}\}$

• Application-Oriented policy

In this policy, the CR with the highest matching level dc_j is selected as the destination of scheduling. The scheduler does not consider the priority levels of tasks, and task classification and prioritization are only performed at the CRs. The strategy can be expressed by the enabling predicate and firing probability of transition d_{ij} in Figure 15. Note that the category and priority of the requests are not explicitly known before they enter the CRs. The enabling predicate of d_{ij} is

$$(M(q_j) < b_j) \wedge (\forall k, 1 \le k \ne j \le m, (dc_{ij} \ge dc_{ik}) \vee (M(q_k) = b_k))$$

The firing probability of d_{ij} is

$$p(d_{ij}) = \begin{cases} |Q|, if \quad j \in Q; \\ 0, otherwise; \end{cases}$$

where Q $Q = \{k | dc_{ik} = \max(dc_{i1}, dc_{i2}, \cdots, dc_{ij}) and M(q_k) < b_k\}$.

This scheduling policy is based on the CR load conditions, and a mechanism is required to dynamically feedback the total number of the tasks of the CRs to the front-end scheduler.

4.5.1.3 Reflective Scheduling Policy

The above two policies are combined to construct the reflective scheduling policy. When there are adequate tasks, the scheduler applies the system-oriented scheme to guarantee a high throughput. When the service capacity of the CR exceeds the number of users' tasks, the scheduler uses the application-oriented scheme to guarantee the best Quality of Service. The term adequate and inadequate could be defined by the scheduler administrator to fit flexible and varying requirements. In this case, the scheduler will switch to the system-oriented heuristic if the number of the available computing resources is less than the tasks. Or it could be switched to the application-oriented heuristic. The reflective scheduling policy can be expressed by the enabling predicate and firing probability of transition d_{ij} in the SHLPN model of Figure 15, where $N_{\max i,j} = b_{ij} + M(q_{ij})$, $N_{\max j} = \sum_{k=1}^{n}(b_{kj} + M(q_{kj}))$

The enabling predicate of d_{ij} according to the System-Oriented Policy is

$$(M(q_{ij}) < b_{ij}) \wedge (\forall k, 1 \le k \ne j \le m, (M_i(q_j) \le M_i(q_k))$$
$$\vee (M(q_{ik}) = b_{ik})) \wedge M(f_i) \le \sum_{j=1}^{m}(N_{\max\ ij} - M(q_{ij}))$$

The firing probability of d_{ij} is

$$p(d_{ij}) = \begin{cases} 1/|Q|, if \quad j \in Q; \\ 0, otherwise; \end{cases}$$

where $Q = \{k | M_i(q_k) = \min(M_i(q_l), M_i(q_2), \ldots, M_i(q_m) and M_i(q_{ik}) < b_{ik}\}$

The enabling predicate of d_{ij} according to the Application-Oriented Policy is

$$(M(q_j) < b_j) \wedge (\forall k, 1 \le k \ne j \le m, (dc_j \ge dc_k)$$
$$\vee (M(q_k) = b_k)) \wedge M(f_i) > \sum_{j=1}^{m} (N_{\max \quad ij} - M(q_{ij}))$$

The firing probability of d_{ij} is

$$p(d_{ij}) = \begin{cases} |Q|, if \quad j \in Q; \\ 0, otherwise; \end{cases}$$

where $Q = \{k | dc_{ik} = \max(dc_{i1}, dc_{i2}, \cdots, dc_{ij}) and M(q_k) < b_k\}$.

4.6 Performance metrics

Let us now consider the criteria which we will use to evaluate the different systems. It is worth noting that different systems have different criteria, whereas the criteria must reflect their main characteristics. It is also evident that the method of evaluating the system is another important factor of judging the system. Performance can be regarded as the most important criteria for

any systems. No matter how powerful the functions that the system can provide, if its performance doesn't reach the criteria it will never be accepted by the users.

For the Grid, the performance is the most important question to be considered. If the Grid is constructed with comprehensive functions, but its speed is very limited, this Grid is a failure. For example when a user submits a task to the Grid, but gets the response one year later, the system definitely eliminates the user to use the system again. From this, we can see that the performance evaluation can be regarded as the guideline to constructing the Grid's infrastructure, the tasks scheduling and resource management.

4.6.1 The criteria of Grid performance

In SHLPN models, the performance measurements can be obtained based on the steady-state probabilities. The metrics used were introduced into this report for the purpose of evaluating the performance. All in all, in those tests, we focus on three parameters, the normal Response Time, the Balance Level and the Efficiency Level. The latter two are carefully designed in the thesis to test the performance of the scheduler.

The throughput of tasks is one of the most important performance measurements. $P[M]$ is the steady state probability for the marking M. The throughput $TH(s_{ij})$ of transition s_{ij} is

$$TH(s_{ij}) = u_{ij} \sum_{M \in E} P[M]$$

where E is the set of markings under which transition s_{ij} is enabled, and the enabling condition is described in the enabling predicate of transition s_{ij}.

The throughput TH_i of the tasks r_i in the Grid system is

$$TH_i = \sum_{j=1}^{m} TH(s_{ij})$$

The throughput TH of the whole cluster is

$$TH = \sum_{i=1}^{n} \sum_{j=1}^{m} TH(s_{ij})$$

Response time of tasks is another important performance metric. In this report, only the response time of the CR, i.e. the elapsed time of a task waiting in the ready queue and being serviced by the CR, is investigated, whereas the transmission time in the networking infrastructure is not considered.

$D(q)$ is the average number of tokens in place q as defined. The response time RT_{ij} by CR transition s_{ij} is

$$RT_{ij} = \frac{D(q_{ij})}{TH(s_{ij})}.$$

The response time RT of tasks r_i is

$$RT_i = \frac{\sum_{j-1}^{m} D(q_{ij})}{\sum_{j-1}^{m} TH(s_{ij})}$$

$V(D(q))$ is the variance of the average number of tokens in place. The balance level for the system is

$$BalanceLevel = \frac{1}{V(D(q_{ij}))}$$

We also consider rejection probability, since tasks rejected first time by a unavailable CR can have a significant impact on performance.

The criterion to evaluate the performance of the Grid is called efficiency level,

$$EfficiencyLevel = \frac{BalanceLevel}{ResponseTime}$$

The reason for using this value as a criterion is because the higher the matching ratio, the more the system can accept tasks from users and also the smaller the response time, the quicker the system can process the tasks. The criterion considers the benefit of both the user and computing resource.

Finally, we use the software package SPNP [28] to obtain the numeric results to our SHLPN model in Figure 15 and explore the performance benefits of Reflective schemes. To avoid the state-space explosion and simplify the model solution, without loss of generality, we only consider a CR consisting of 5, 10, 15, 50 and 100 computing resources.

4.6.2 Static Environment Test

We assume the number of the CRs are fixed at the value m. These service rates are assumed to have been known beforehand by gathering the statistics. The arrival rates of tasks are 250 and 2500 tasks per second separately.

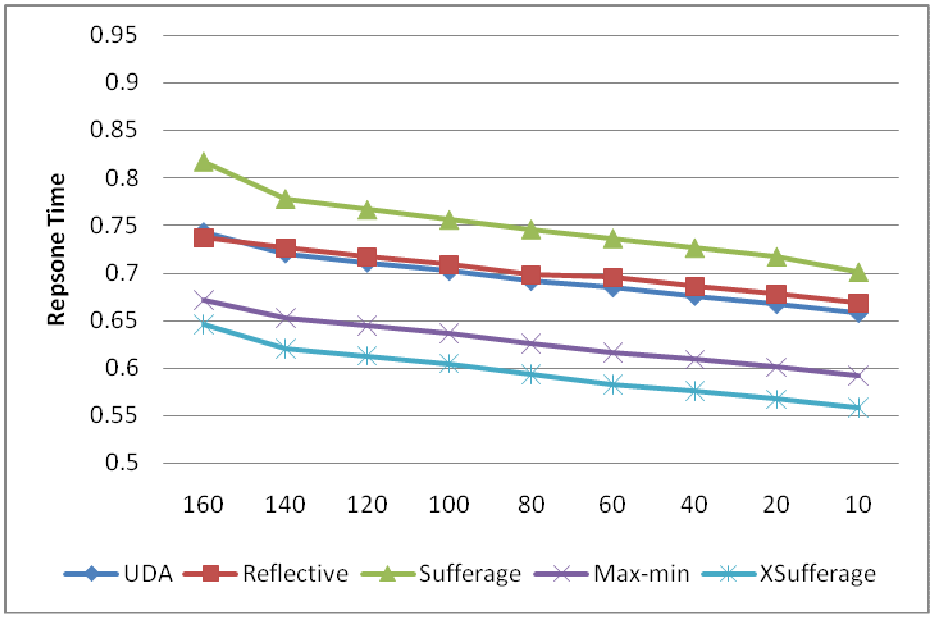

Figure 16: the Response time for different scheduling scheme for 250 tasks per second with different number of available computing resources

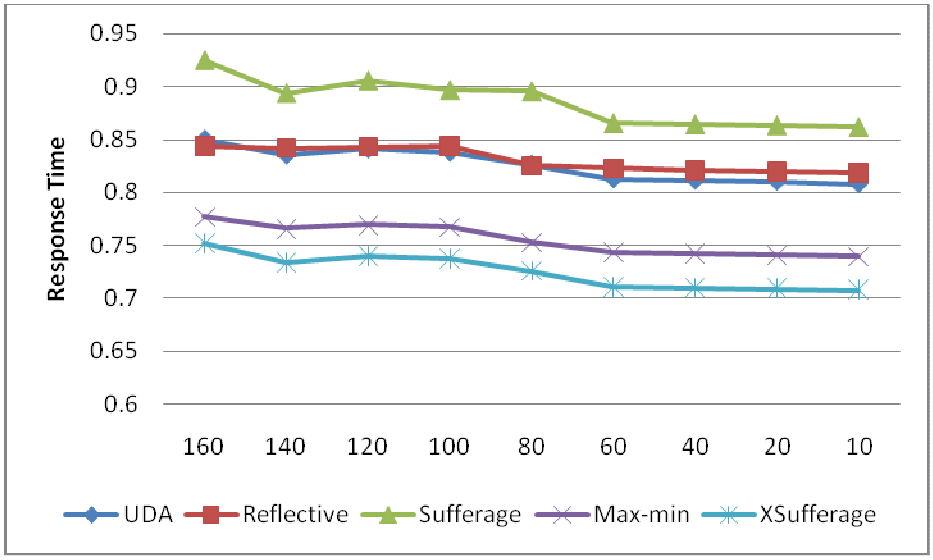

Figure 17: the Response time for different scheduling scheme for 2500 tasks per second with different number of available computing resources

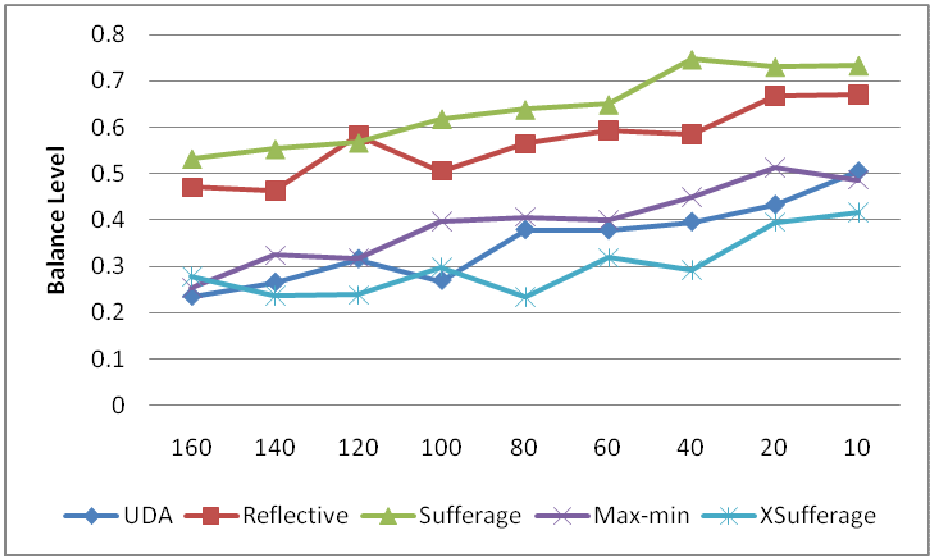

Figure 18: the Balance Level for different scheduling scheme for 250 tasks per second with different number of available computing resources

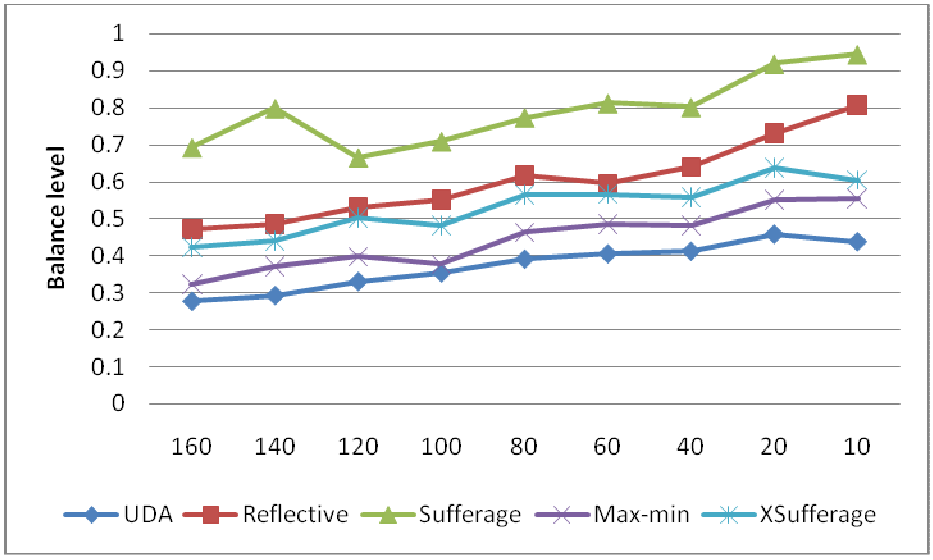

Figure 19: the Balance Level for different scheduling scheme for 2500 tasks per second with different number of available computing resources

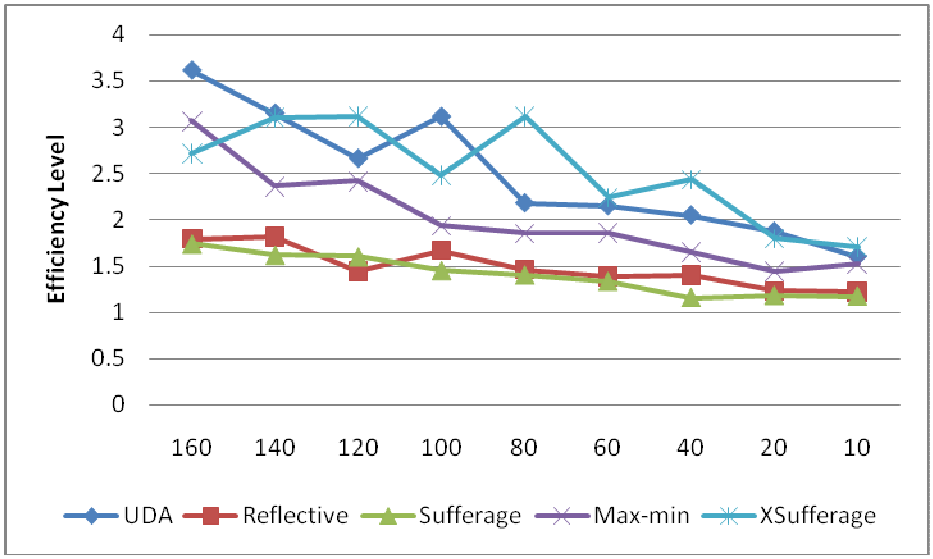

Figure 20: the Efficiency level for different scheduling scheme for 250 tasks per second with different number of available computing resources

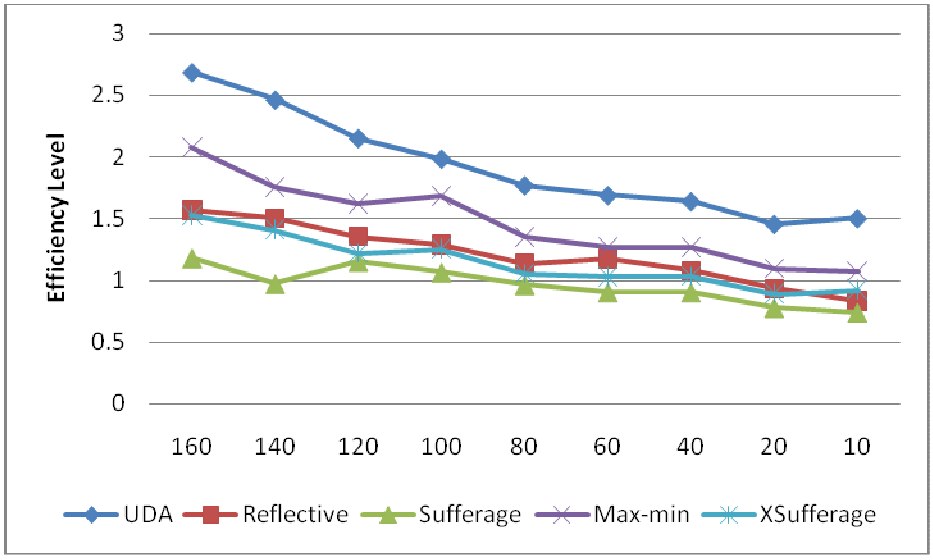

Figure 21: the Efficiency level for different scheduling scheme for 2500 tasks per second with different number of available computing resources

From the figure, the performance of the Reflective heuristic performs in quite a mediocre manner. For the test with 250 tasks per second, the Reflective is almost the worst one. For the test with 2500 tasks per second, the Reflective performs in the middle of other heuristics.

Ranking	250 tasks per second	2500 tasks per second
UDA	1 or 2	1
Reflective	4 or 5	3
Sufferage	4 or 5	5
Max-min	3	2
XSufferage	1 or 2	4

TABLE 2: RANKINGS FOR DIFFERENT HEURISTICS UNDER DIFFERENT TASK ARRIVAL RATES

These results show that the reflective heuristic is not ideal for a static or slowly changing environment as it is designed for the environment when the computing resources dynamically change. The reasons for this come mainly from two areas:

- The System-oriented heuristic or Application-oriented heuristic itself is not the optimal heuristic given an environment with no changes.
- The forecasting function becomes a time-consuming module and not a useful option in this case. Furthermore, the calculation of statics may generate noise which is unacceptable in the static environment.

4.6.3 Dynamic Environment Test

In the dynamic environment, we assume that the computing resources can be dynamic joining or leaving the Grid. We assume that computing resources join or leave the Grid with a probability with Normal distribution. The reason for

use of the Normal distribution here is because it can express the probability of a number of events occurring in a fixed period of time if these events occur with a known average rate and independently of the time since the last event. Therefore in the previous model, the number of the computing resources is no longer a constant, which means the value m obeys the Normal distribution. In this case, the greater the variance of the Normal distribution, the more volatile the system is. Therefore in this section, we display results with different.

The simulation changes the variance of the Normal distribution, from 0.1 to 2. It is worth noting that the variance value is measured by the mean of the task length. This can roughly reflect that the Grid is from a less dynamic environment to a very volatile system.

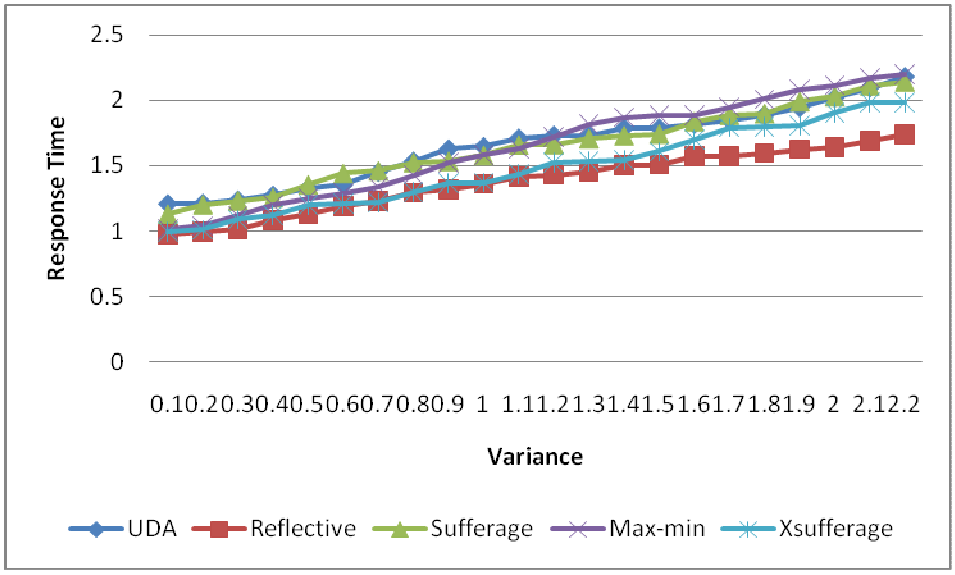

Figure 22: the response time for different scheduling scheme for 250 tasks per second with different 100 number of available computing resources dynamic joining or leaving the grid

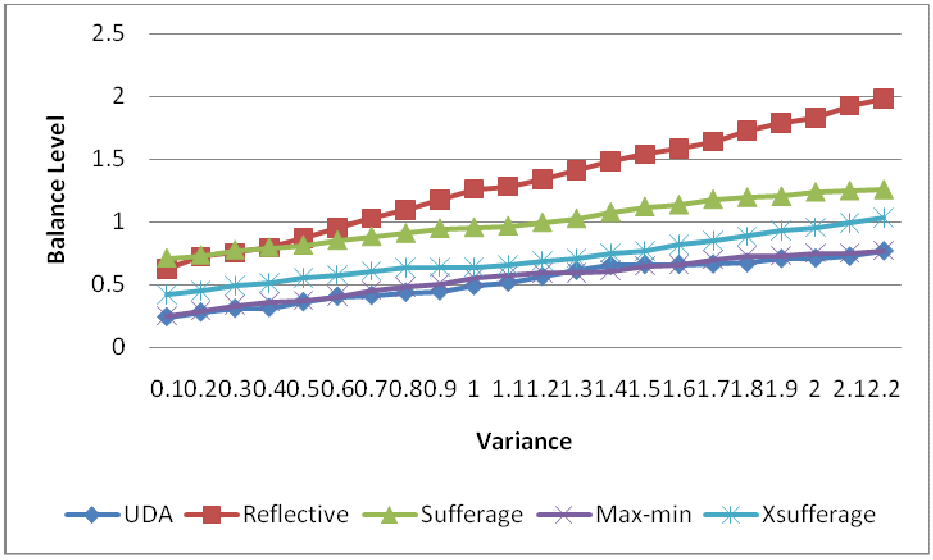

Figure 23: the balance level for different scheduling scheme for 250 tasks per second with 100 computing resources dynamic joining or leaving the grid

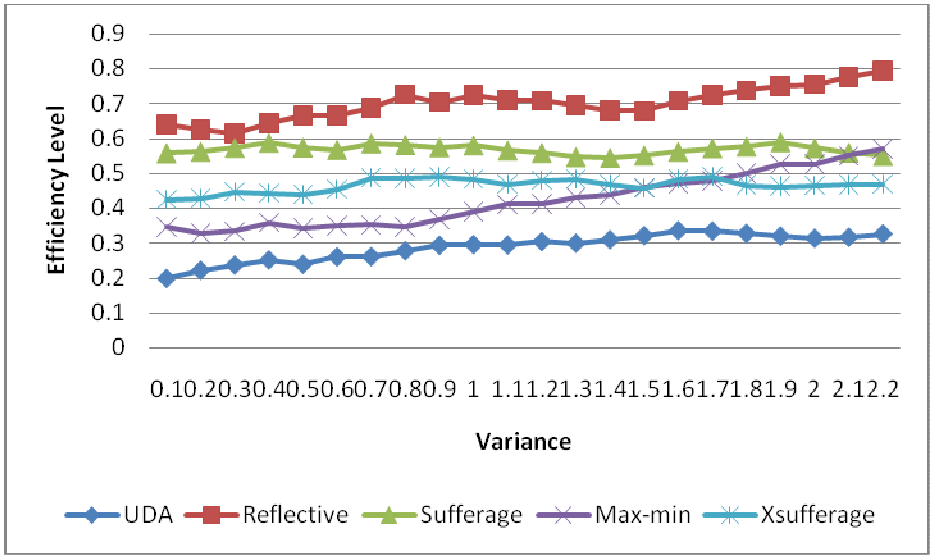

Figure 24: the Efficiency Level for different scheduling scheme for 250 tasks per second with 100 computing resources dynamic joining or leaving the grid

The above figures describe the efficiency level for different heuristics. Unsurprisingly, the Reflective scheme achieves the best performance results.

The reason for this is that it can find the balance between the load balance and the user's satisfaction. The simulation result is that the Reflective heuristic outperforms other heuristics in the dynamic environment. The more volatile the environment is, the better it performs. For the test with variance 0.1, the Reflective scheduler outperforms the second best heuristic, Sufferage, by about 13.6% in respect of the Efficiency Level. For the test with variance 2, the Reflective outperforms the second best heuristic Max-min by about 36.9% in respect of the Efficiency Level. This suggests that the Reflective heuristic can contribute a robust and satisfactory performance compared to other heuristics.

Variance of the Normal Distribution	Percentage	Variance of the Normal Distribution	Percentage
0.1	0.136676	1.1	0.228716
0.2	0.106423	1.2	0.238056
0.3	0.068807	1.3	0.240098
0.4	0.091106	1.4	0.22251
0.5	0.148121	1.5	0.208347
0.6	0.159793	1.6	0.232106
0.7	0.160623	1.7	0.236786
0.8	0.223432	1.8	0.248017
0.9	0.203614	1.9	0.327829
1.0	0.222571	2.0	0.369403

TABLE 3: THE PERCENTAGE THAT REFLECTIVE SCHEDULER IS BETTER THAN SECOND BEST SCHEDULERS IN RESPECT OF EFFICIENCY LEVEL

Summary

In this chapter, we firstly discussed three different evaluation approaches for the Grid, mathematical model, stimulation, meteorology and so on. Then the

mathematical approach was picked out to measure the performance of the system. Because the key idea of this approach is based on the queuing theory, in the following sub-section we briefly described queuing theory and its implementation model Petri Nets. Later chapter, we used Petri Nets to describe five different scheduling schemes, Sufferage, XSufferage, reflective, max-min and UDA. With different numbers of tasks, the models give us some meaningful results for different schemes.

In summary, the Reflective heuristic worked in a very mediocre manner in a static environment, due to the design of the System-oriented and Application-oriented heuristics and the forecasting module. However, as we discussed in Chapter 3, the reflective scheme outperformed the other four schemes when the environment is constantly changing.

Chapter 5 System Design

Following the above chapter, the general structure of the scheduler will be represented. It gives more detailed information about how the scheduler actually is constructed. The task partition can be created by using skeleton library tools, or generated automatically with the aid of Grid information services such as MDS and VDS prior to the run time. A workflow specification defines the workflow and the data dependency of tasks. At run time, the reflective scheduler manages the execution of the workflow by utilizing Grid middleware. There are three major components in a main scheduler: the reflective scheduler, data movement and fault management. Reflective scheduling discovers resources and then allocates tasks onto suitable resources to meet users' requirements. Data movement manages data transfer between selected resources, and fault management provides mechanisms for failure handling during execution. In addition, the scheduler provides feedback to a monitor so that users can view the workflow process status through a Grid workflow monitor.

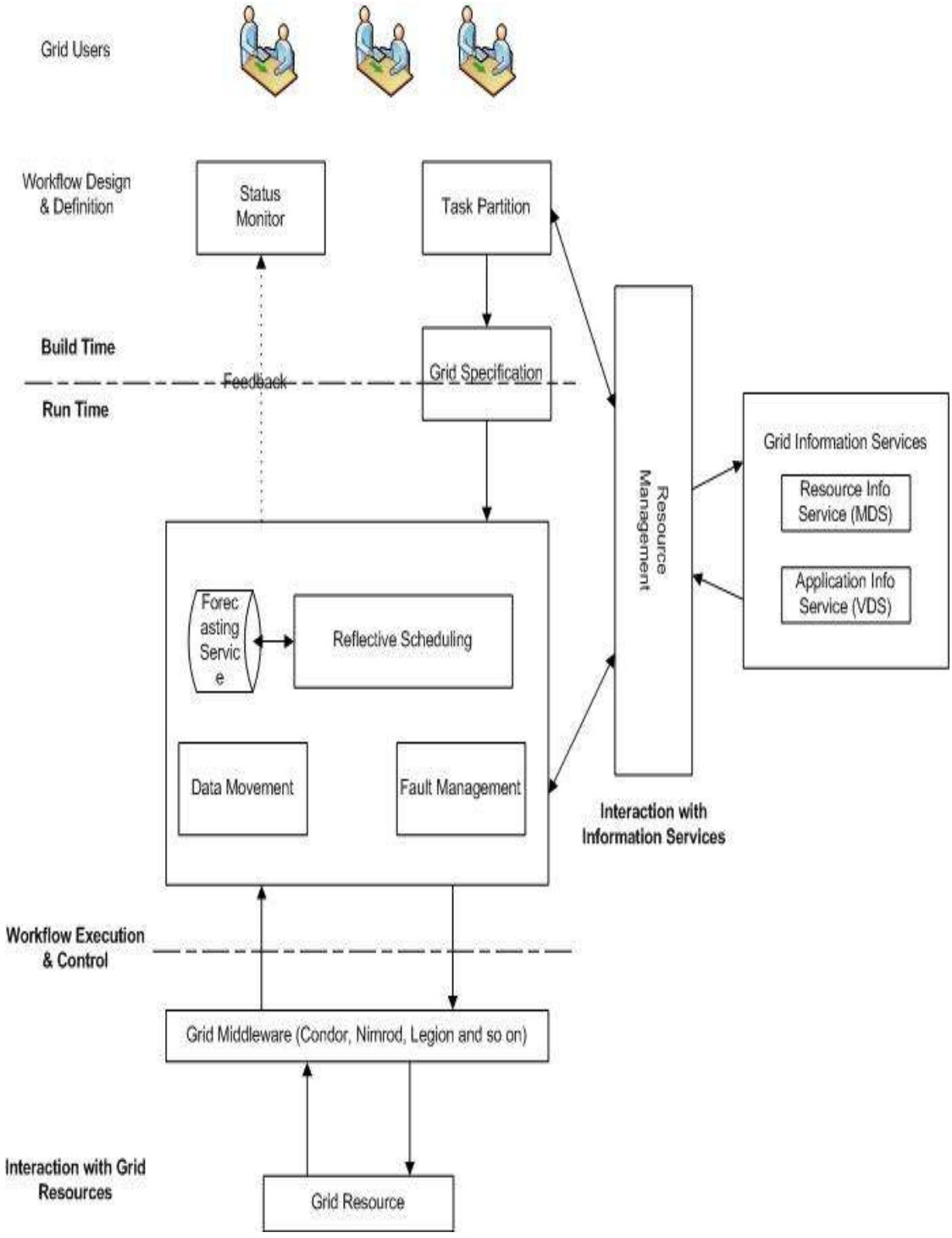

Figure 25: The general structure of the reflective scheduler describing basic modules in the scheduler

5.1 Scheduler Design

While grid computing is not aimed specifically at the high performance computing community, many organisations regard it as a means to deliver commodity computation that exceeds the capabilities of their current systems.

In a static environment, application performance may be improved through the tuning of an appropriate algorithm, the re-mapping of data, or the adjustment of communication behaviour, all of which can be considered, to some extent, during development. In a dynamic grid setting, system-wide issues such as resource configuration, user contention, node failure and congestion can have a major impact on performance and are difficult factors to predict a priori, particularly when access to system information is restricted. Additionally, computational providers will invariably offer services to users with conflicting needs and requirements. Contract issues including deadline and response times must be balanced in order to meet respective service-level agreements.

It is very important for a workload management system to allocate suitable resources for a specific task, given an open resource pool. Here, by open, it means varied and dynamic. So a just-in-time approach is employed in this research where performance models of an application are evaluated as tasks are submitted, allowing the system to consider run-time parameter prior to execution.

5.2 Scheduler Structure

The scheduler engine shown in Figure 25 is the core part of the design. It is composed of seven major tools: Task Analysis Tool (TAT), Task Classification Tool (TCT), Resource Analysis Tool (RAT), Resource Classification Tool (RCT), Mapping Tool (MT), Performance Prediction Tool (PPT) and Allocation Analysis Tool (AAT).

Task Analysis Tool is used to analyse the task's source codes and data, then describes the task by the task specification language (TSL) scripts. The scheduler also allows the users to submit their tasks with the prepared scripts. When the user submits the task, the scheduler first checks whether it has the TSL scripts: if not, it will send the task to the Task Analysis Tool.

Task Classification Tool is used to classify the tasks. If the users submit the tasks without the task category TC, it will use the TSL scripts to analyse the tasks and then put them in a suitable task category. If the users submit the tasks with TC, it will classify the task by this parameter. All the tasks will be put into the Task Category Table according to the TC. For example, the task needing high volume storage will be classified into the Data Repository; the task needing high performance computing will be categorized into the High Performance Computer.

Resource Analysis Tool is similar to Task Analysis Tool, but it is designed for the resource side. It is used to help the scheduler analyse the resource's specification. After the analysis, the Resource Analysis Tool uses the RSL

scripts to describe the resources and stores all the information in the Resource Database. Any changes in the resources noticed by the broker will result in an update in the Resource Database.

Resource Classification Tool is used to classify the resources. If the resources submit the resource information without resource category RC, it will use the CSL scripts to analyse the resources and then put them in a suitable resource category. If the resources submit the resource information with RC, it will classify the resource by this parameter. All the resources will be put into the Resource Category Table according to the RC. For example, the resource with high volume storage could be classified into the Data Repository; the task with high performance computing power will be classified into the High Performance Computer. The Resource Classification Tool will modify the resource's category according to its performance during the Grid environment.

MT is used to map the task to the resource. First using the information from the Task Category Table and the Resource Category Table, the MT will map the task to the relevant resource category. The task belonging to Data Repository will be mapped to the resource in the Data Repository category in an ordinary condition. If there is no resource which could satisfy the user-defined requirement in this category, the MT will map this specific task to all the resources.

Performance Prediction Tool will give the forecasting information about the next period and check whether the information reaches the triggering level.

When it reaches the triggering level, the Performance Prediction Tool will notify the Allocation Analysis Tool to change the current scheduling heuristic.

Allocation Analysis Tool is the most important tool in this design, because it will allocate the specific task to the most suitable resource. Using this tool, the scheduler will consider the resource's usage efficiency. In this step, two heuristics could be used. First is the Application-oriented heuristic, which can find the most effective resource to fit the user's task. The second is the System-oriented heuristic, which is similar to the RAM management scheme in Linux [66]. The Application-oriented heuristic assigns the resource with highest MA value to the tasks in order to guarantee the user's satisfaction. The MA value is used to evaluate the quality of the resources, which will be discussed in detail in a later chapter. The System-oriented heuristic assigns the resource with the lowest workload to the tasks in order to provide a better load balance in the system.

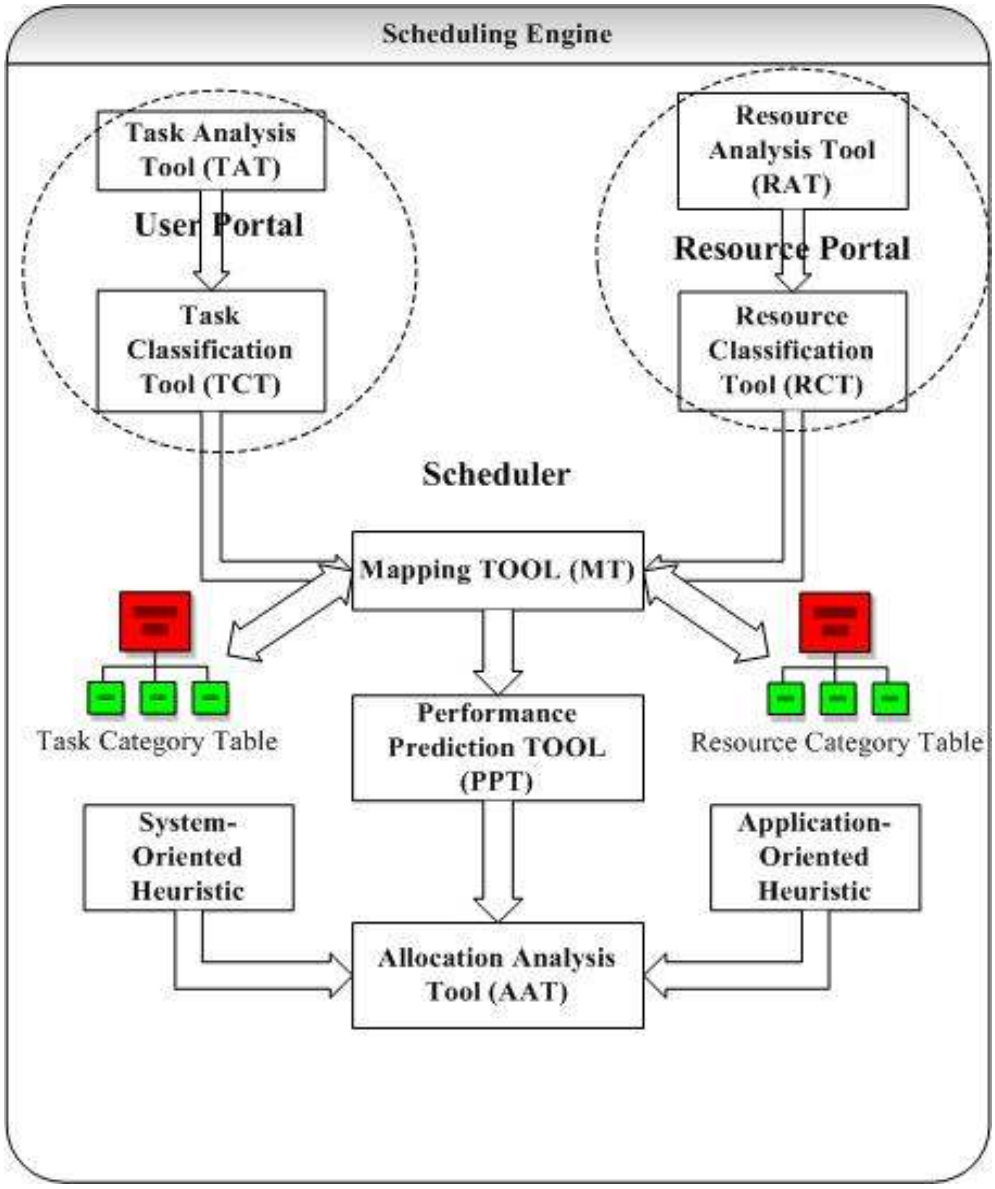

Figure 26: the details design of the reflective scheduler which gives the information about each module

5.3 Scheduling Representation

Before starting the task execution procedure, we will introduce two parameters which are key factors affecting our design: MA and BA. MA is used to judge the matching level of the resource and the task. The smaller the

MA value is, the higher the matching level. BA is used to represent the adequate level between the tasks and resources.

MA Calculation: Considering the resource's burden and usage efficiency, the system's AAT (Allocation Analysis Tool) will calculate the matching value for all the pre-qualified resources. The resource with the smallest matching value is regarded as the most suitable one under this scheme.

When talking about whether the computing resource is suitable or not, it is impossible to bring in any qualitative based parameters which are hardly being used by the computing logics. Thus finding a way to represent the suitability of the resource becomes critical. In other words, it is vital to find a formula to explain the matching level of a certain resource and a certain sub-task.

While the users submit the task, they need to specify the requirement for their sub-tasks. So there will be a vector to display the user's requirement:

$$R = [r_1, r_2, r_3 ..., r_{n-1}, r_n]$$

Furthermore, the users need to specify the importance level of each part:

$$I = [i_1, i_2, i_3 ..., i_{n-1}, i_n]$$

For example, in one resource there are five components in all: CPU, Storage, Networking, RAM and Graphic. If this resource has a very large volume of storage and the administrator treats the resource as a data repository, the

cost element regarding the Storage could be set to be a very low value. Compared with another resource with the same CPU, Networking, RAM, Graphic and smaller Storage, for the first task with very large data to store, although both resources could meet the user's requirement, the suitable one could be the first resource. Also for each computing resource, there is a vector describing their specifications:

$$Specification = [s_1, s_2, s_3 ..., s_{n-1}, s_n]$$

So the matching level can be calculated as:

$$MA = (S - R)I^T$$

Where I^T stands for the transposed matrix of I

Generally speaking, the higher the MA value, the more suitable the resource. However, there are some exceptions which need to be discussed:

- Most of the computing resources' specifications will be ranked from 0-10 in the resource category table. For example, the system sets 3 GHz CPU speed as 10 and 300 MHz CPU speed as 1. The same for the memory size, storage volume, network speed and so on.
- Some special equipment such as scanner, photocopier and printer will only be ranked 0 or 10, where 0 stands for not available and 10 stands for available.

- Normally, the equipment will not be that critical. Unless the task's requirement is critical, for example the printer is essential, then we set the value to be 10.
- The table describes possible specifications that may need to be considered in the future if the scheduler considers the heterogeneity of the tasks :

Type	Possible Parameters
Job Flow	Parallel -> Networked -> Serial
Different Jobs	Single Job-> Multiple Jobs
Sub-jobs Depth	No Subjobs-> Deeply staged subjobs
Job Types	Batch-> Simple-> Parallel-> EJBS based jobs-> Complex jobs
OS dependent	Independent-> Strongly depending
Memory size needed	Small-> Large->
DLL in place	Standard DLLs-> Specific DLLs
Compiler Setting	No compiler-> Standard settings-> Special setting
Runtime environment	None required-> Standard runtime-> Runtime required
Application Server	None Required-> Simple beans/JSP-> EJB-> Specific needs
Foreign Application	None Required-> Standard application-> Special settings/installation
Hardware dependent	None-> Standard IT devices-> Special IT devices-> Special other devices
Redundant Job execution	Not Required-> Heavily Depending on
Scavenging Grid	All Jobs individualized for scavenging-> Not suitable for scavenging
Job data I/O	Command line parameter-> Message queue-> Data file-> Database-> APIs->
Shared data access	RO files-> RO DBMS-> RW files-> RW DBMS->

Temporary data space	Small->nearly unlimited (check out concurrent jobs on each node)
Network bandwidth	Small->High speed network LAN->WAN
Time-Sensitive Data	Data always valid->Time depending data values
Data type: Character sets	Commonly available Unicode in SBCS network ->Different Unicode in DBCS->Inconformity of character codes on network
Data type: Multi-media formats	Uniform use of set of multimedia formats->Mixed use of formats
Time constraints	No time restrictions apply->Strong need for timely execution and data provisioning
Migration needs	Grid in fixed environment->Grid application based on common standard->Grid likely to migrated on different platforms
Data separable per job	Data easily Separable->Some solvable data interdependencies->Data inseparable
Amount of Data	Small amount of I/O data per job->Large amount of data handled by single jobs
Job topology	Simple job topology (job-node-data)->Complex job topology
Data topology	Simple data topology(data-job-node)->Complex data topology
Network scalability	High upper limit in scalability graph->Low upper limit
Single User Interface	Not required->Standard UI->Integrated common UI

TABLE 4: THE PARAMETERS INCLUDED IN THE TASK SPECIFICATIONS

Furthermore, the resource's component may have different costs, so adding the cost weight to the above function on one hand could improve the resource's usage efficiency based on its own condition. On the other hand, it could give the scheduler a clear definition about the resource's type (its advantages and disadvantages). The cost matching function could be:

Where the $Cost = [c_1, c_2, c_3 ..., c_{n-1}, c_n]$ represents the cost for the component per unit time, it will be updated by the resources. Any policy changed and components updated will result in the modification of this parameter. Thus, the allocation policy of the scheduler will be changed sequentially.

BA Calculation: BA value is another key fact affecting the reflective mechanism. It reflects the balance between the resources and the tasks, while Task Number/Resource Number>=BA value, there are adequate tasks left, and vice versa.

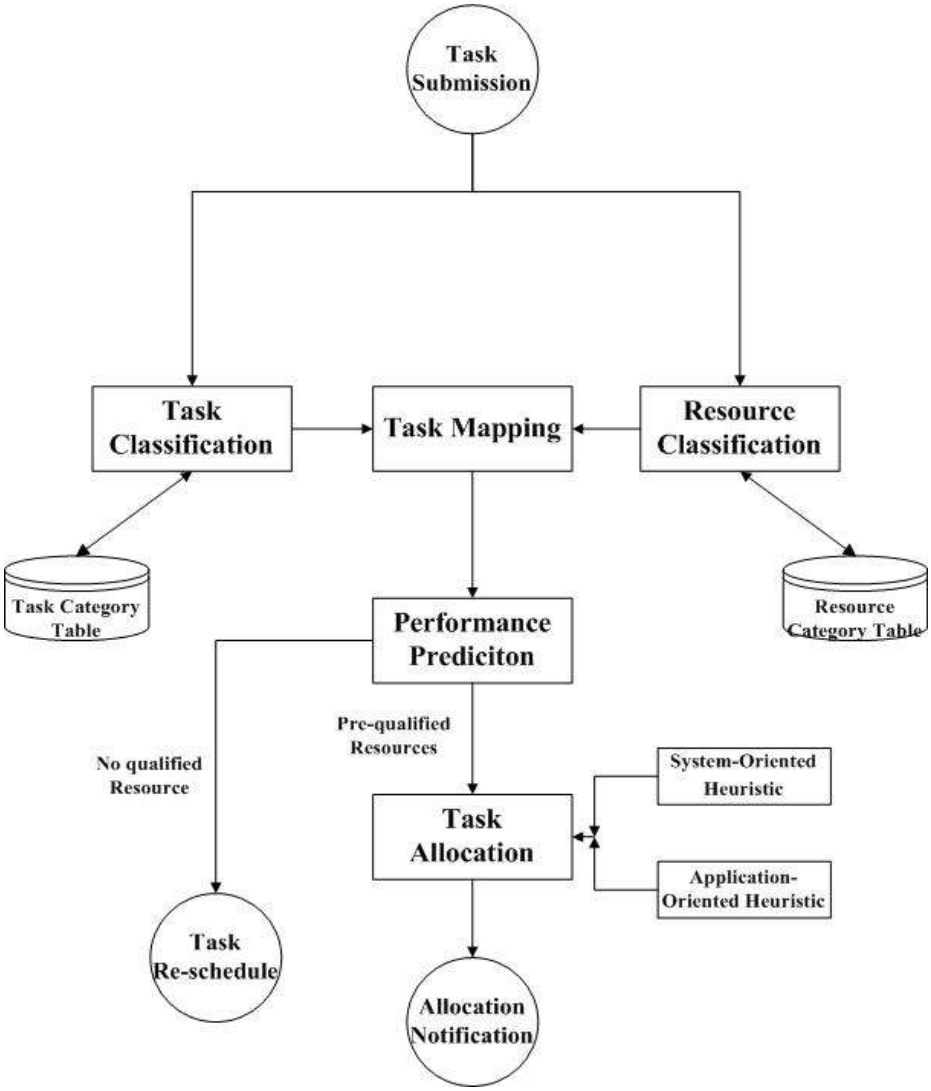

Figure 27: The workflow of the reflective scheduler from Task Submission to Allocation

There are five steps shown in Figure 27 from the task submission to task allocation: Task Submission, Task Analysis & Resource Analysis, Performance Prediction, Allocation Analysis and Task Allocation Notification.

Step1 Task Submission: The users submit their task with the maximum waiting time Tmax, Task Category TC and/or TSL (Task Specification

Language) scripts. The Task Category TC is used to describe the task's type, such as High Performance Computing or Large Memory Computing. The TSL is used to describe the task's specifications and execution requirement.

Step2 Task Analysis & Resource Analysis: If the task is submitted without the TSL scripts, the Task Analysis Tool will start to work and give a TSL script. At the same time, the Resource Analysis Tool will check the information to find whether there is any update in the resources. If it finds any, the Resource Analysis Tool will modify the RSL scripts in the Resource Database.

Step3 Task Classification & Resource Classification: The Task Classification Tool will classify the tasks using the TC parameter, or if the tasks are without this parameter, it will analyse the TSL scripts and then use the analysis result to classify the tasks. A similar action is used for Resource Classification Tool . Initially, the Resource Classification Tool will classify the resources according to their RT. Then for a fixed period of time, the Resource Classification Tool will send the enquiry to all the resources within the Grid environment and any resource which receives the enquiry will report the changes. After this, there will be a modification for all the resources. Another condition is that if the resource in any category has not been used for a long time (reaching a specified value), the Resource Classification Tool will re-classify the resource.

Step 4 Performance Prediction: Using the Blume adjustment approach and historical information, the scheduler will predict the Grid situation for the next period.

Step 5 Global Mapping: The MT will allocate the task to the resource with the matching category for the first time. If the MT receives the notification from the Performance Prediction Tool, it will map this specific task to all other resources.

Step 6 Task Allocation: First the Task Analysis Tool will ask the scheduler administrator to set the maximum MA value MAmax and then set the Balance value BA. In this step, there are two schemes from which the Task Analysis Tool can choose. The first one is the System-oriented heuristic.

Step 7 Task Re-schedule: Suppose there are three sub tasks divided from a meta-task, if the first one has been completed as well as the third one, the second sub task is regarded as having failed. This task is then put into the high priority queue to reschedule.

Summary

In this chapter, the thesis presents the details of how to implement the reflective scheduler. The first part of the chapter talks about the structure of the scheduler, their functionalities and how these modules are used in a real working environment. Then the thesis gives the logic behind the structures and explains the reason.

Chapter 6 Virtual TestBed Project

In this chapter, the project called VTB (Virtual TestBed) will be introduced. Moving on from the theoretical stage, this thesis also puts the reflective scheduler theory into practice and then tests the system with a real application. This application is used to solve the geometry optimisation problem. The original version is standalone or non-parallelized, which was developed by Bo Xu [107]. In the latter part of this chapter, the thesis will show how to distribute this application across a Grid infrastructure using the reflective scheduler.

6.1 Introduction to the geometry optimisation application

The application is designed to solve geometry optimization problems. In order to run the calculation, several core modules are needed:

- Representation of the geometry

  The shape of any geometry will be described by "*two cubic B-spline curves with nominal uniform knot set*" [107]. Each curve is then defined by a number of control points. The more control points, the more precise the

curve is. However, more control points means more time consuming by the calculation.

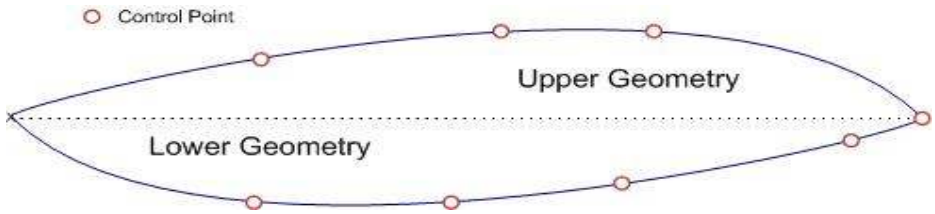

Figure 28: Sample splines to construct the closed geometry with 8 control points

- Mesh generator

  A mesh generator is needed to produce mesh information for any suitable geometry. Again, the number of triangles inside the geometry will decide the precision of the calculation and, again, the more triangles contained in the geometry has, the more time is consumed by the calculation.

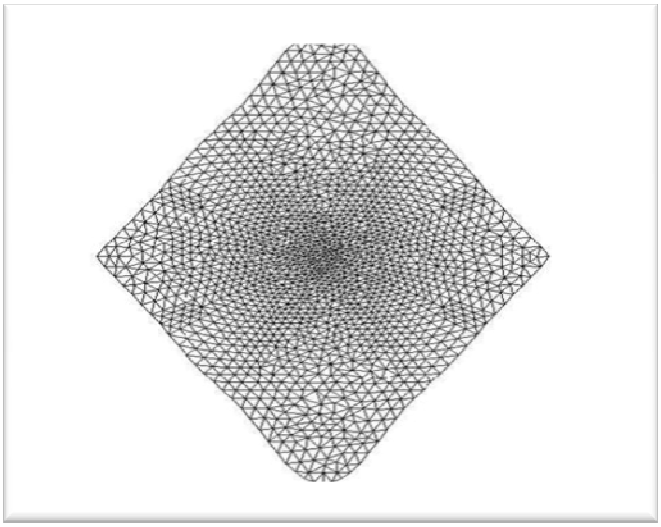

Figure 29: Sample mesh produced by the mesh generator with 10000 triangles

- Poisson solver

  A PDE solver is used to solve the Poisson equation using certain meshes.

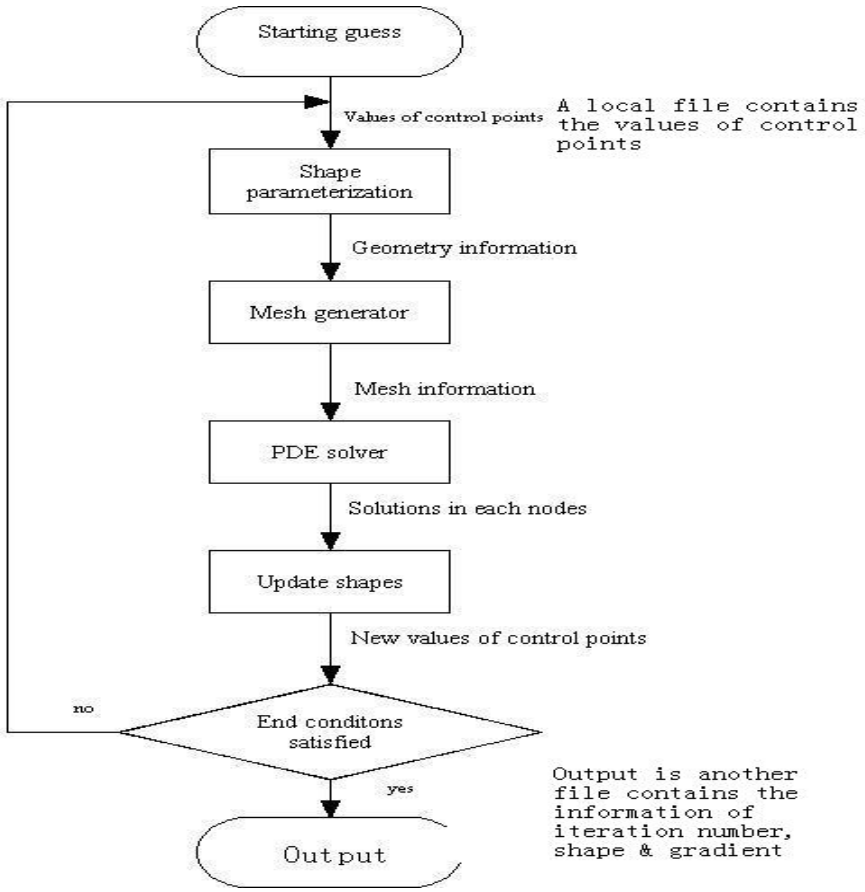

Figure 30: The logical structure of the shape optimisation application [107]

6.2 Detailed Design

From the structure, we can see that there are mainly three modules constructing the program. After investigation, the module PDE solver is found that can be easily parallelized [27]. Also, we know from [107], in order to decrease the total computing time, it is possible to "*compute the gradient*

vector in parallel or grid computing if the finite difference method is used". As the calculation for each control point is independent, so the application can employ the Farm skeleton from the Skeleton Library (see Appendix). The flow chart below explains how these items are used in the optimization scheme.

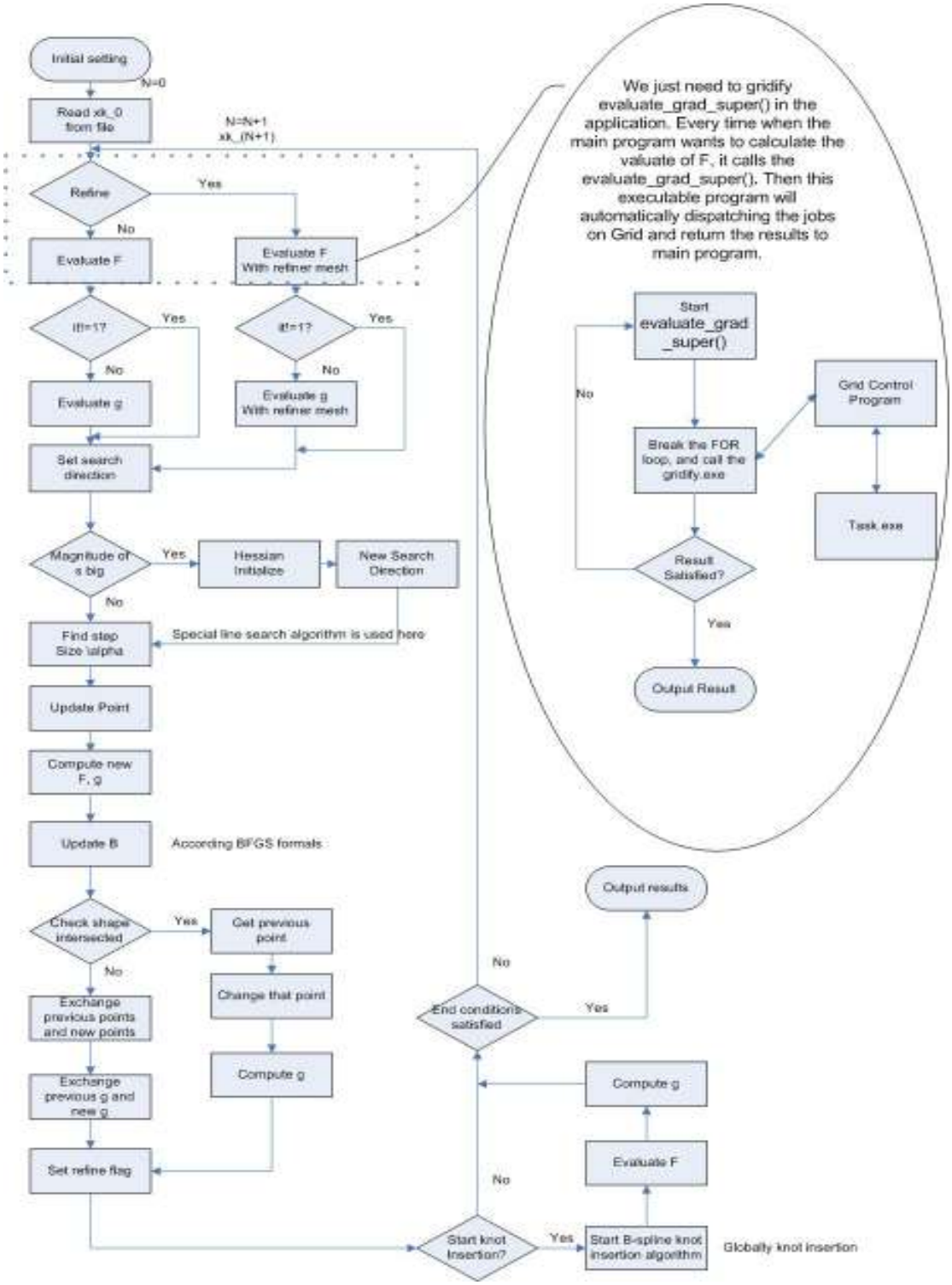

Figure 31: the workflow of how the application is partitioned and dispatched to a grid through the scheduler

In order to schedule this application on the Grid, two applications are needed:

- One called ProgramMaster as the interface for users. It is used to help the user to submit tasks to the Task Allocation Tool of the scheduler.
- One called ProgramSlave as the interface for the Grid. It is used to help the Allocation Analysis Tool of the scheduler to submit tasks to the gatekeeper of the Grid.

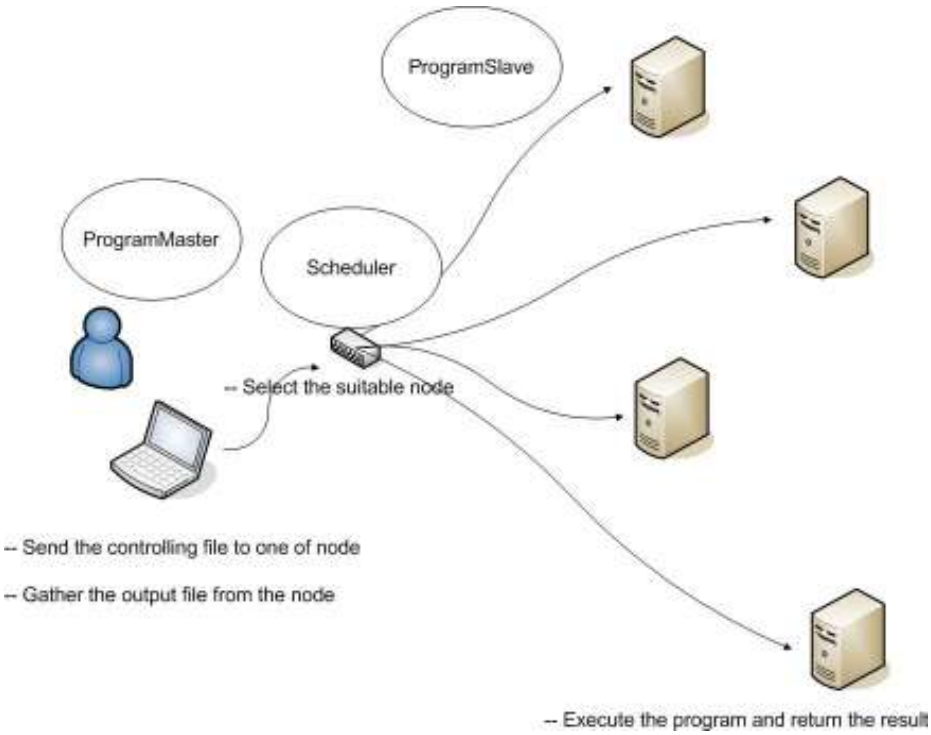

Figure 31: The logical structure of how the application is partitioned and dispatched to grid through the scheduler

One problem is that the application uses a control file and returns one output file that needs to be passed to, and retrieved from, a remote host. Inter-process communications between the nodes cannot be used. One solution is

to serialize the results by storing them on local disks and using the Globus Toolkit 2.2 data movement functions:

- By using GRAM [30] and GASS [15] systems , t he file will be transferred to the selected remote host.
- By using GRAM and GASS systems, the results of the execution node will be output on the master node.

Communication is accomplished:

- By a local GASS server started on the master node and listening on port 10000
- By the GASS server started on the execution node by GRAM, which will map standard input and output to remote files.

The following RSL command describes this process:

&(executable=ProgramSlave) (arguments=<controlling file>)

(stdout=https://<masternode>:10000/<localdir>/output file)

(stdin=https://<masternode>:10000/<localdir>/controlling

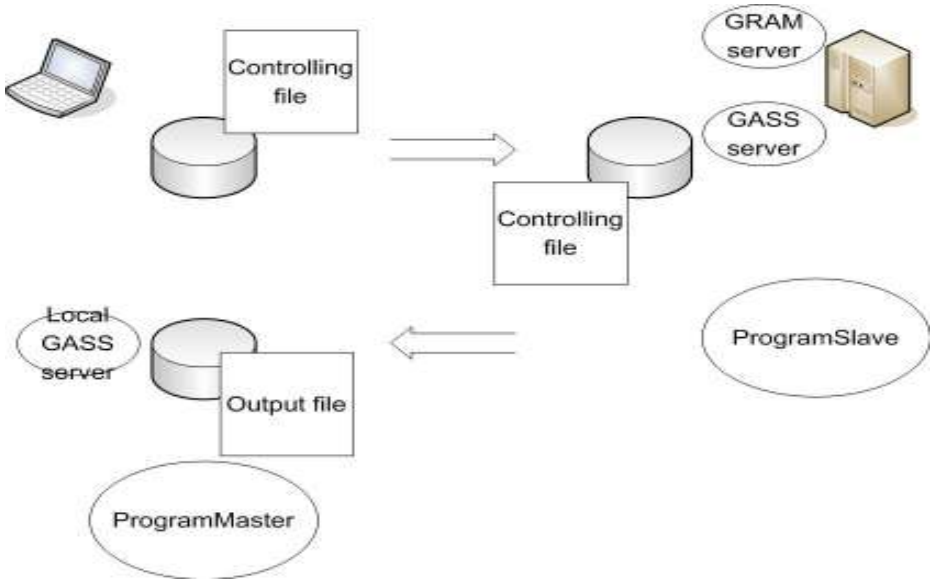

Figure 32: the process of how the application is partitioned and dispatched to grid through the scheduler

The local GASS server started by ProgramMaster transparently provides access for the controlling file to the remote execution node via GRAM and the associated GASS server. It listens on port 10000 (an arbitrarily chosen free port). Two functions StartGASSServer() and StopGASSServer(), wrap the Globus Toolkit 2.2 API calls to start and stop the local GASS server. The ProgramSlave shows that it reads and writes only on standard input and output channels (via *cin* and standard *iostream* objects). It will nevertheless transparently work on remotely stored files. Finally, the GridFtp protocol will be used to transfer the ProgramSlave executable to a remote host.

In this thesis, four experiments have been carried out, each with different configurations:

- In the first one, the application runs on a single computer without the Grid platform. In other words, it is under the original sequential environment.
- In the second one, the application runs on the Grid platform with two computers, one as scheduler and one as pure computing resource.
- In the third one, the application runs on the Grid platform with three computers, one as scheduler and two as pure computing resources.
- In the fourth test, the application runs on the dynamically changing environment with different scheduling heuristics. The environment is trying to simulate the real Grid environment, in which computing resources join or leave the infrastructure.

In order to simplify the problem and ease the analysis, in these experiments, we assume that the computers' performance is exactly the same and without any networking problems? delays. The specification of the computing resources is:

- Intel® Pentium® 4 Processor 2.0G Hz
- 1Gb RAM
- 80GB 7200rpm IDE Hard Disk
- 100MB Motherboard with onboard LAN adapter
- Globus Toolkit 4.0
- Microsoft Windows XP professional edition

6.3 Testing the efficiency of the skeleton library

The first stage is to verify whether this parallelization by using the skeleton library can reduce the execution time. An understanding of each stage of the computing process will help to further optimize the system.

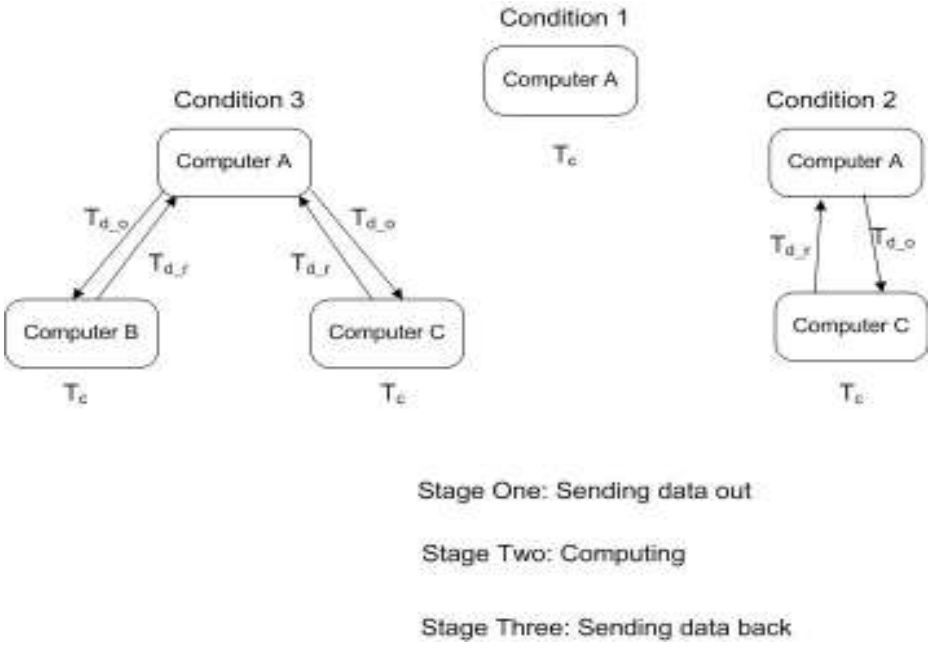

Figure 33: Comparison among the tasks running on a single computing resource, two computing resources and three computing resources

T_n represents the total time from the beginning of the Grid-enabled task to the end of the task, from the time: SYSTEM CALL to the time: RETURN RESULT;

T_{d_o} represents the time used to transfer the data files, for the time: DISPATCH

T_{d_r} represents the time used to transfer the data files back, for the time: COLLECT RESULTS

T_c represents the time used to calculate the data

T_e represents the time taken for the application to collect the final result, and for the time left in part in the evaluate_grad_super();

T_f represents the time taken for the scheduler to get the return results from the return data files, for the time: COORDINATE SUB-RESULT and the time: SYSTEM CALL.

N represents the number of control points.

p represents the number of pure computing resources in the Grid platform

In the first configuration, as there is no Grid environment involved, so the application carries out N times' iterations. The total time for the fat is therefore:

$$T_0 = N \times T_c + T_e$$

In the second arrangement, we use the Grid platform with only one pure computing resource. In this experiment, the computing time is the same. However, for each loop, the application will cost the time in data file transfer, data collection and data synchronization. In this case we have:

$$T_1 = N \times (T_c + T_{d\_o} + T_{d\_r}) + N \times T_f + T_e$$

Analogously, we can get the following equation when there are two or p pure computing resources and one scheduler:

$$T_2 = \frac{N\times(T_c+T_{d\_o}+T_{d\_r})}{2} + \frac{N\times T_f}{2} + T_e$$

……..

$$T_p = \frac{N\times(T_c+T_{d\_o}+T_{d\_r})}{p} + \frac{N\times T_f}{p} + T_e$$

In this condition, if the computer's performance is unchanged, the T_0, T_1 and T_e are fixed, so we obtain:

$$T_c = \frac{T_0-T_e}{N}$$

$$T_{d\_o} + T_{d\_r} = \frac{T_1-(N\times T_f+T_e)}{N} - T_c = \frac{T_1-(N\times T_f+T_e)}{N} - \frac{T_0-T_e}{N} = \frac{T_1-NT_f-T_0}{N}$$

$$T_p = \frac{N\times(T_c+T_{d\_o}+T_{d\_r})}{p} + \frac{N\times T_f}{p} + T_e = \frac{T_1-NT_f-T_0}{p} + \frac{N\times T_f}{p} + T_e = \frac{T_1-T_0}{p} + T_e$$

When $N>1$

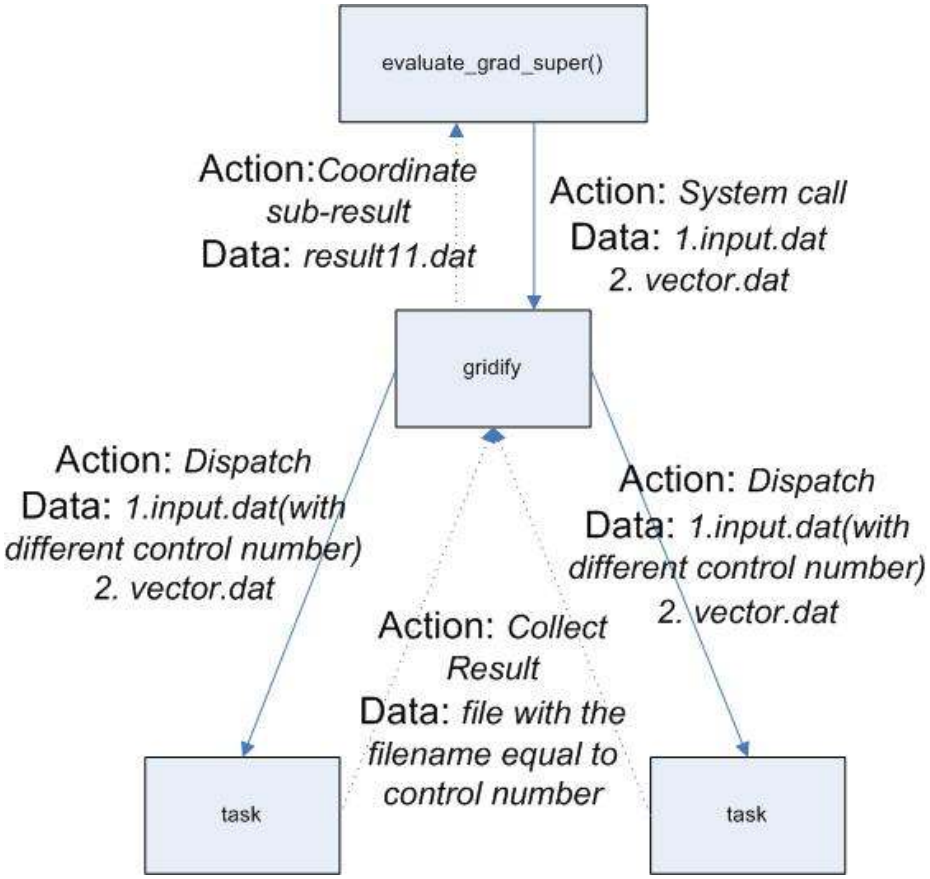

Figure 34: Detailed description about how the application is partitioned and dispatched to grid through the scheduler

Control Point=20 Triangles=10000	Total Time	Extra Time	Computing Time	Data Transfer Time	File Time	Theory Time
a single computing node	69s	4s	3s	0s	0s	64s
One as scheduler; One as a computing node	310s	6s	3s	10s	1s	292s
One as scheduler; The other two as pure computing nodes	192s	6s	3s	12s	1s	166s

TABLE 5: THE EXECUTION TIME FOR EACH PART OF THE CALCULATION FOR ONE ITERATION WHEN THE NUMBER OF CONTROL POINTS IS 20

Control Point=38 Triangles=10000	Total Time	Extra Time	Computing Time	Data Transfer Time	File Time	Theory Time
a single computing node	251s	6s	6s	0s	0s	234s
One as scheduler; One as a computing node	713s	7s	7s	10s	1s	691s
One as scheduler; The other two as pure computing nodes	395s	9s	7s	11s	1s	370s

TABLE 6: THE EXECUTION TIME FOR EACH PART OF THE CALCULATION FOR ONE ITERATION WHEN THE NUMBER OF CONTROL POINTS IS 38

From Table 5 and Table 6, we have changed the control points from 20 to 38. This modification increases percentage of the computing time in the total

execution time. We can find that the theoretical estimate of the execution time is close to the actual execution time. This validates our timing models. The more time consumed in computation, the better performance is obtained by parallelization. From Table 5, the three-node structure is 2.59 times as long as the one-node structure in the total execution time. However, from Table 6, the three-node structure is 1.59 times as long as the one-node structure. The improvement is obviously due to the computing time increases a a percentage of the total time.

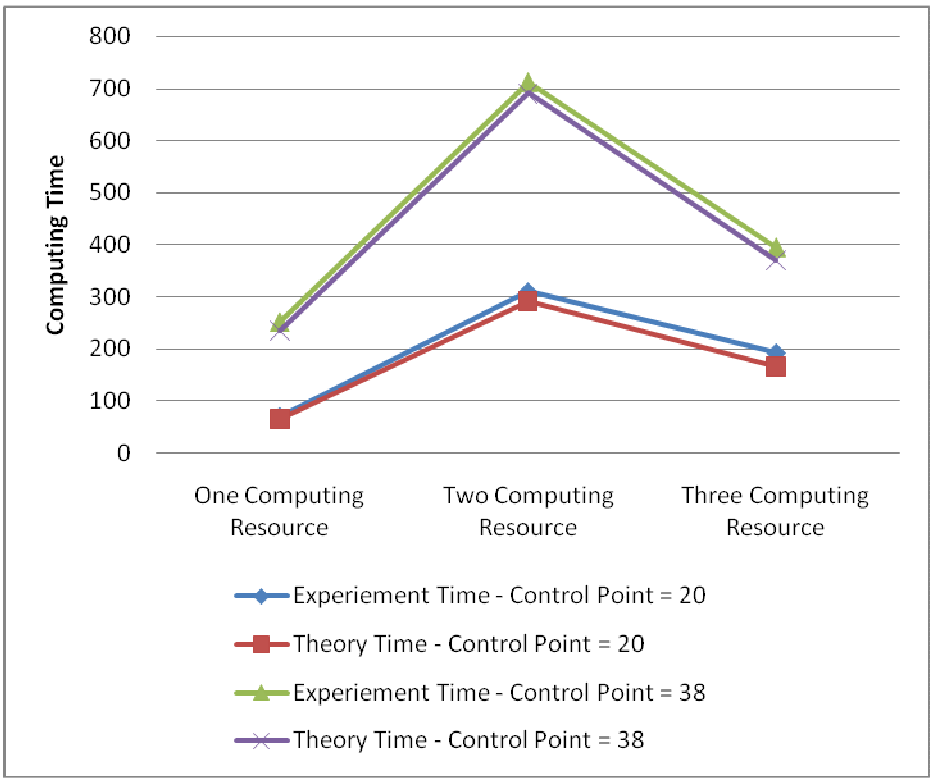

Figure 35: the computing time for the application by using one, two and three computing resources

Structure	Triangles =10,000	Triangle =8,000	Triangle =6,000	Triangle =4,000	Triangle =2,000
a single computing node	6s	5s	3s	0.8s	0.7s
One as scheduler; One as a computing node	7s	5s	3s	1.2s	1s
One as scheduler; The other two as pure computing nodes	7s	6s	3s	1.4s	1s

TABLE 7: THE COMPUTING TIME WITH DIFFERENT NUMBERS OF TRIANGLES IN ONE ITERATION

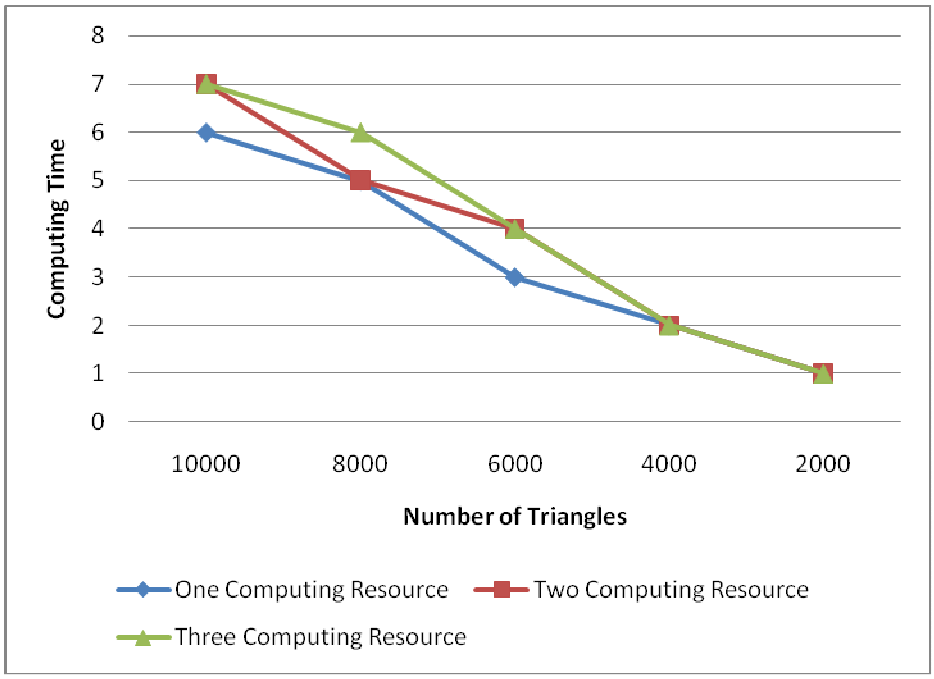

Figure 36: the computing time for the triangles with one, two and three computing resources

In Table 7 and Figure 36, we can see that as the number of the triangles increases, the computing time increases accordingly. Therefore, the percentage of the computing time increases.

Structure	Triangle =10000	Triangle =8000	Triangle =6000	Triangle =4000	Triangle =2000
a single computing node	271s	223s	197s	117s	88s
One as scheduler; One as a computing node	713s	653s	601s	572s	453s
One as scheduler; The other two as pure computing nodes	395s	341s	285s	197s	143s

TABLE 8: THE TOTAL CALCULATION TIME FOR ONE ITERATION WITH DIFFERENT NUMBERS OF TRIANGLES

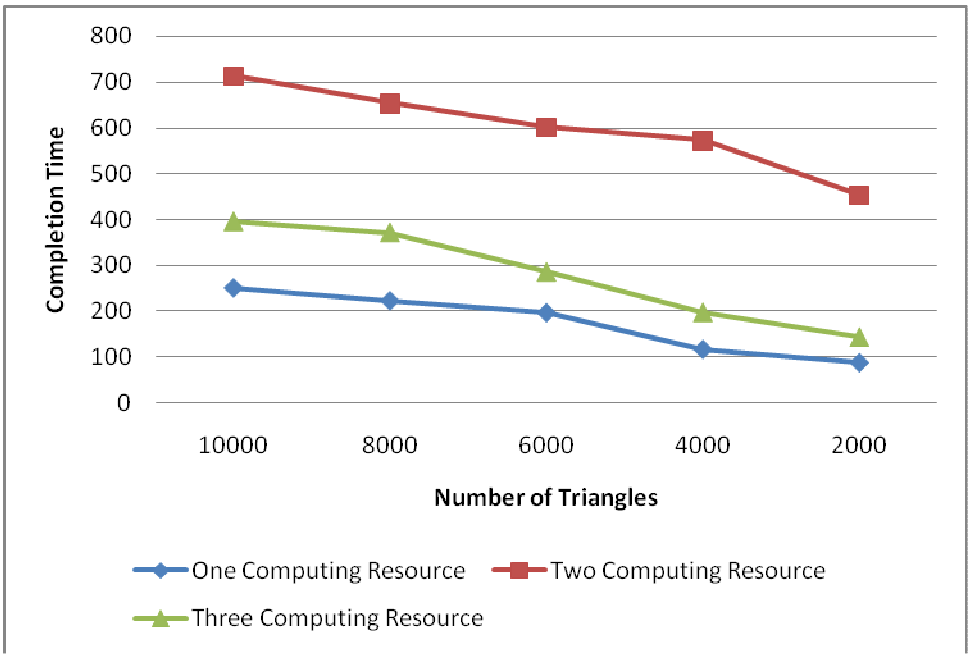

Figure 37: the number completion time for the triangles with one, two and three computing resources

According to both to Table 7 and Table 8, the modification in the number of triangles increases percentage of the computing time in the total execution time. The more time consumed in computation, the better the parallel performance. In Table 8, the three-node structure is 1.62 times as long as the one-node structure in total execution time when the number of triangles is 2,000. However, the three-node structure is 1.46 times as long as the one-node structure when the number of the triangles is 10,000. The improvement is obviously due to the computing time increases as a percentage of the total time.

6.4 Testing the efficiency of reflective scheduler

The final stage is to test this application based in the real Grid environment which allows the computing resources to dynamically join or leave the environment. In this test, we used 38 computers with the same specifications:

- Intel® Pentium® 4 Processor 2.0G
- 1Gb RAM
- 80GB 7200rpm IDE Hard Disk
- 100MB Motherboard with onboard LAN adapter
- Globus Toolkit 4.0
- Microsoft Windows XP professional

Each computer was running Virtual PC 2004[73] which could setup a virtual machine locally with different specifications. In this test, all the specifications are randomly chosen. Therefore a heterogeneous environment can be easily setup. Hence Grid environment can be provided by using Virtual PC

The following figure shows the distribution of available computing resources during a day. This reflects the dynamism of the Grid environment.

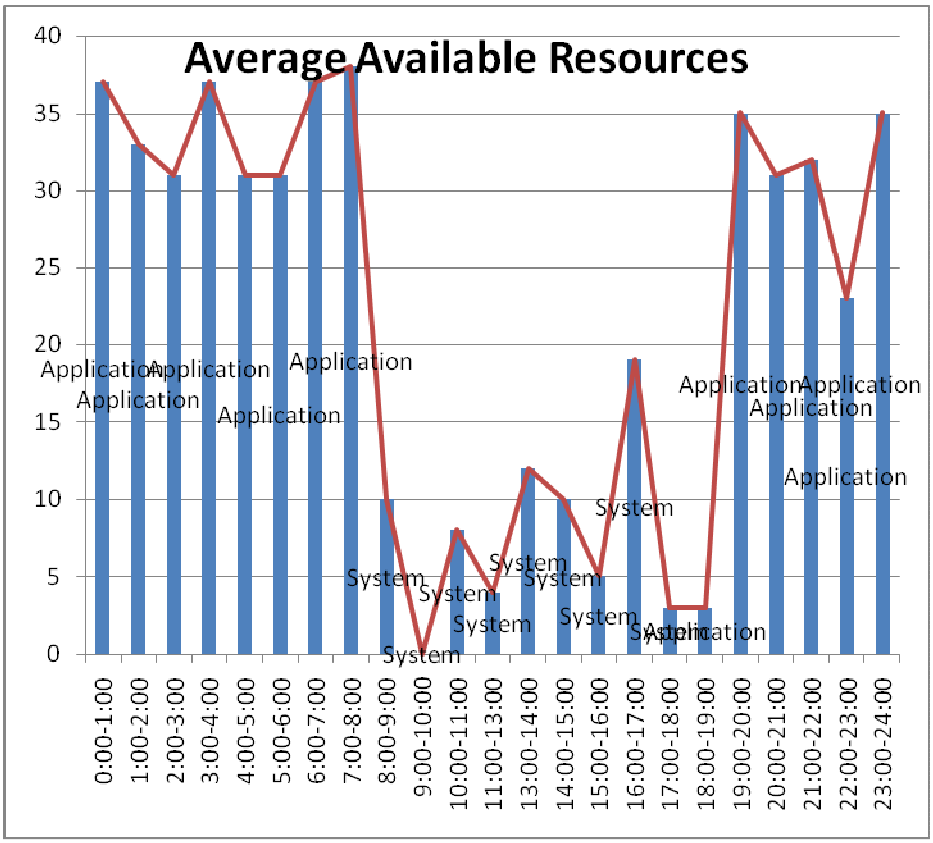

Figure 38: the distribution of the available number of computing resources during a day

Also the standalone version is executed on a 880 b. A supercomputer which runs the Sun Solaris (Operating System) (the machine comprises, 72,

ultraSparc 1.2G Hz processors, 288GB shared memory). To make a meaningful comparison, the experiment only uses 38 processors.

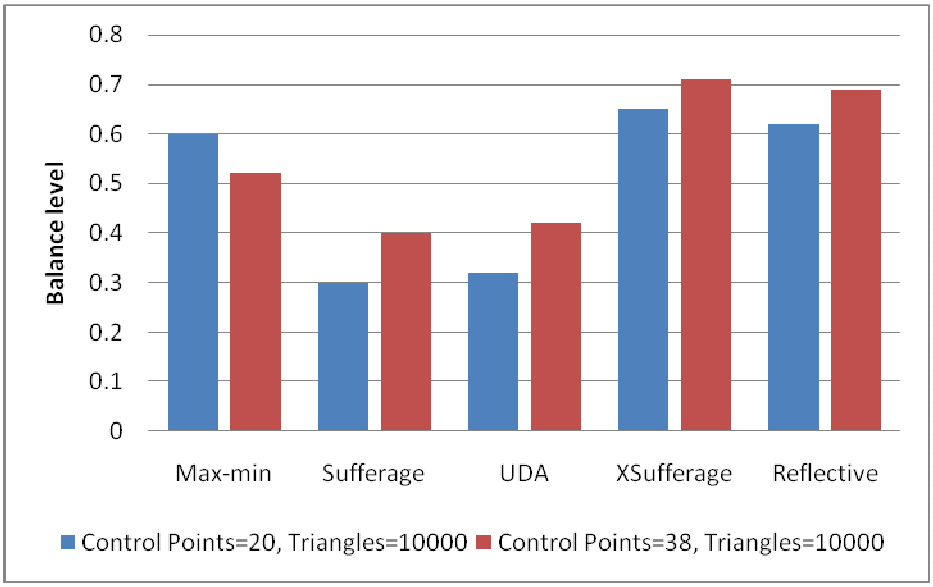

Figure 39: the balance level by using different scheduling heuristics

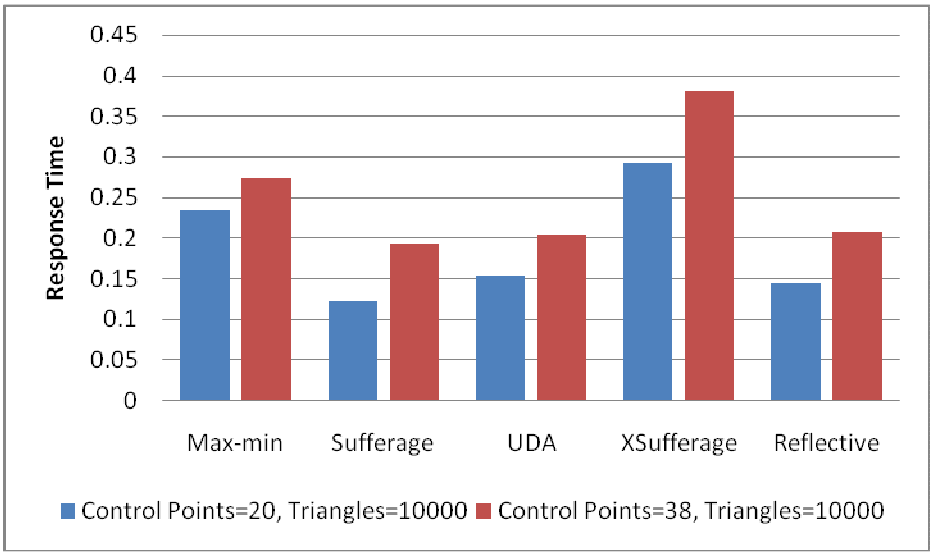

Figure 40: the response time by using different scheduling heuristics

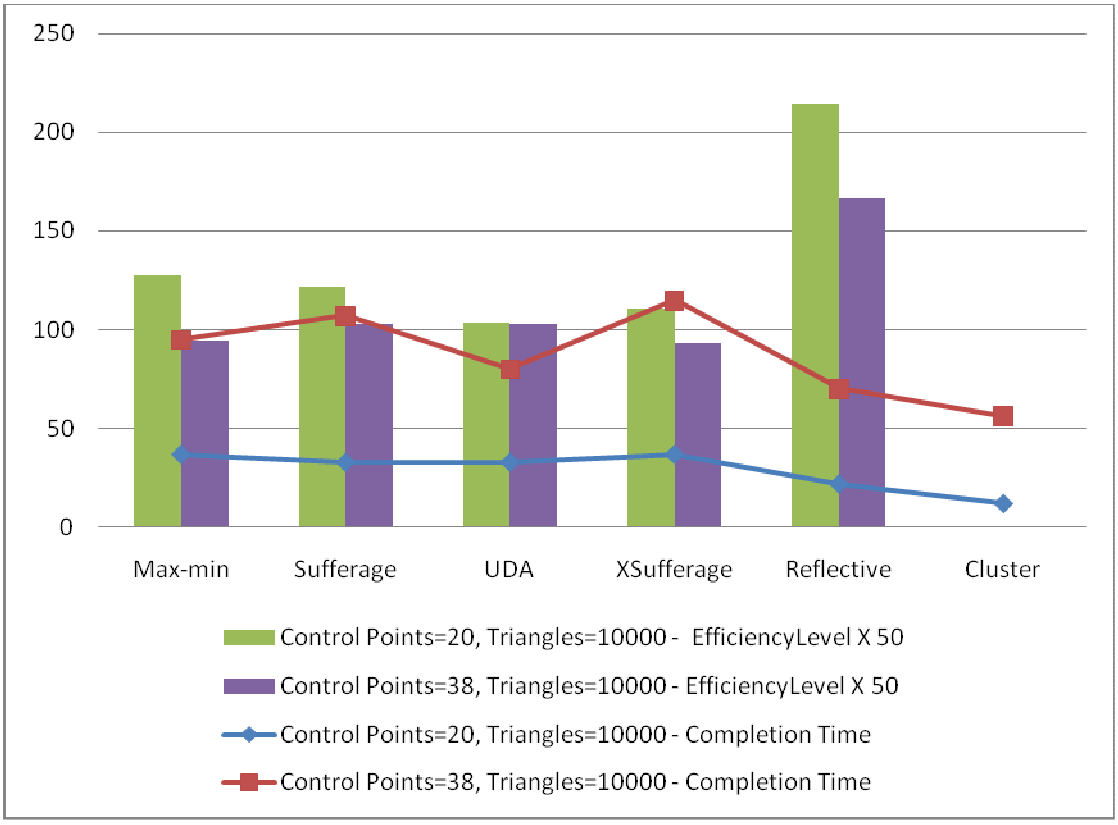

Figure 41: the completion time and efficiency level by using different scheduling heuristics

In this test, there are several interesting things which are worthy of note:

- The Efficiency Level can be regarded as a critical criterion to judge the performance of the Grid. In Figure 50, the results show that the higher the Efficiency Level, the lower the completion time.
- Using our new scheduling heuristic significantly increases the performance of the Grid. For the calculation with 10,000 triangles and 38 control points, the *reflective scheduler* can reduce the completion time by about 39% compared to the second best heuristic, Max-min. For the calculation with 10000 triangles and 20 control points, the *reflective scheduler* can reduce

the completion time by about 21% compared to the second best heuristic, Sufferage.

- Compared to the performance of the cluster, the reflective scheduler can achieve very promising performance. For the calculation with 10,000 triangles and 38 control points, the reflective scheduler's completion time is only 19% slower compared to the cluster. For the calculation with 10,000 triangles and 20 control points, the Reflective scheduler's completion time was only about 11% slower compared to the cluster. According to the formula above, the Grid can outperform the cluster given that the number of computing nodes over 70. Furthermore, comparing the cost of the cluster and the Grid, the Grid is almost free as it takes advantage of the idle computing resources.

Summary

In this chapter, one of the large calculation applications was implemented into the reflective system. During the first stage, the researcher tested the theory that using the *farm structure* gridification skeleton could reduce the completion time with at least three or more?? more computing resources. This application was then applied to the real Grid environment in which the configuration of the system dynamically changed. The reflective heuristic outperforms other scheduling schemes in the completion time. It can outperform the second best (the Max-min heuristic) in the test with control points 38 and triangles 10,000

by about 39% in respect of the completion time. It also can outperform the second best application-oriented heuristic in the test with control points 20 and 10,000 triangles by about 21% in the completion time. Therefore, this clearly indicates that the reflective design and the skeleton library not only simplify the Grid task design, but also significantly improve the performance.

Chapter 7 Conclusions and future work

Resource management and scheduling systems in Grid environments need to be flexible to handle dynamic changes in the availability of resources and user requirements. At the same time, they need to provide scalable, controllable, measurable, and easily enforceable policies for the management of resources. Conventional resource management systems are very efficient at managing static and dedicated resources for high-performance computing, but cannot handle the complexity and dynamism of Grid environments efficiently.

Furthermore, the Grid provides an easy way to construct a very powerful computational structure, allowing the user to run parallel programs. In order to maximize the usage of the Grid, there will be potential requests to the scheduler to provide task partitioning functionality. With this functionality, the single meta-task can be easily divided into a number of sub-tasks to be executed with remote resources.

To address these requirements, in the thesis we have proposed a novel scheduling infrastructure for resource management and scheduling, which

essentially consists of two core components to support the requirements stated above.

The idea of *reflectiveness* is the core of this design, which gives the scheduler the ability to change its scheduling algorithms. Reflectiveness means that the scheduler can adjust its scheduling schemes according to different trigger conditions dynamically measured in the Grid. The embedded triggering condition in our system is based on the comparison of the number and the type of user tasks with the number and types of computing resources. And embedded scheduling schemes are application-oriented and system-oriented. However, our framework also allows users to develop their own scheduling heuristics according to different triggering conditions. This system is very flexible; it is well adapted to future rapidly varying Grid environments.

Skeleton programming helps the scheduler control the workflow of the tasks as well as providing a solution to support the scheduler divide tasks into sub-tasks.

The measurement of performance is another important criterion for our system. In these studies, we use a mathematical tool: Petri-Nets to describe our system and its behaviour and we run several simulations using this model. Following this, we applied the model to a real application: the Virtual TestBed project, where we used the Grid environment to calculate geometry optimization problems. The test results from both simulations and real

applications suggest the reflective design and the skeleton library not only simplifies the Grid task design but also significantly improves the performance.

7.1 Contributions

In summary, the work we have done is original, and improves the performance of the Grid compared to other schedulers.

Our contributions to the field of resource management and scheduling systems include:

- In Chapter 3, the thesis describes the reflective mechanism and a scheduler based on this principle. Traditional schedulers, which mostly use market-based heuristic, have limitations in dynamic Grid environments. Therefore, in this thesis, the reflective concept uses welfare economic theory to construct a more practical and efficient Grid market. The idea of reflectiveness can be used to account for the dynamism of the Grid environment. Although other related works provide an adaptive solution, they are only focused on parameter sweep applications. The idea from welfare economics about the multi-objective optimisation enhances the predictive and planning capabilities of the scheduler. Finally, to support the reflective heuristic, the scheduler needs to estimate the condition of the Grid in the next period. A statistical method called Blume adjustment has been proposed in order to forecast the condition of the environment. In

that test, for 0:00 to 6:00, the error of the forecasting is less than 8% and in the worst case, the error of forecasting is less than 15%. This test shows that there is indeed a predictable trend in the ratio of the available resources. Using this information, the system can generally “understand” the conditions for the next period. Due to the sample size of the historical information obtained, the performance can be improved in the future once the system has stored longer history.

- The task partitioning support function described in Appendix. The original idea of the skeleton library came from the eSkel project [29], which is used for the parallelization of programs on clusters. This thesis has developed this idea and put it in the Grid environment in order to simplify Grid programming. In this chapter, the thesis first reviews advantages of using the skeleton programming technique in traditional parallel computers. Then it proposes a way to take advantage of such a technique in the Grid environment. Followed by this, the thesis employs two skeletons: Farm and Pipeline. Finally, a test is implemented to compare the performance of the library in a Grid environment and MPI in a cluster environment. In the Farm test, when the number of the computing resources is small, say up to 8, the speedup in the cluster is almost twice as much as the Grid. When the number of compute nodes is increased to 24, the speedup in the cluster is about 20.8% better than in the Grid. Again, in the Pipeline test, when the number of the computing resources is small, say up to 10, the speedup in the cluster is almost twice as much as the Grid. When the

number of the computing resources is increased to 24, the speedup in the cluster is about 15% better than in the Grid. The reason for this is quite obviously that the cost of the communication is too high and cannot be offset by the parallelization. The more computing resource the Grid uses (provided the problem is sufficiently large), the smaller the impact of the overhead. When the number of the Grid is large enough, say up to 20, the performance of these two are very close. Therefore, we can say that the skeleton library can provide an efficient and convenient way for the user to partition the tasks.

- The Petri-net simulation in Chapter 4 uses queuing theory to model the Grid environment by Stochastic High Level Petri Net and simulate the model by using the Stochastic Petri Net. In this chapter, we first discussed three different evaluation approaches for the Grid, mathematics model, simulation, meteorology and so on. Then the mathematical approach was picked out to measure the performance of the system. Because the key idea of this approach is based on queuing theory, in the following sub-section, we have briefly described queuing theory and its implementation model -Petri Nets. In a later chapter, we used Petri Nets to describe five different scheduling schemes, Sufferage, XSufferage, reflective, max-min and UDA. With different requirements, the models gave meaningful results for different schemes. In summary, the *reflective* heuristic worked in a very mediocre fashion in a static environment, due to the design of the System-oriented and Application-oriented heuristics and the forecasting module.

However, as discussed in Chapter 3, the reflective scheme outperformed the other four schemes in a changing environment.

- In addition, we applied the new scheduler to a real research problem: Virtual TestBed (for geometry optimisation calculations) in Chapter 6. The researcher has applied the scheduler into a real project called Virtual Testbed in order to decrease the computing time. In this chapter, one of the large calculation applications has been implemented into the reflective system. In the first stage it was tested using the farm structure Grid-enabled skeleton which could reduce the completion time with at least three more computing resources. Then this application was applied to the real Grid environment, in which the configuration of the system is dynamically changing. The reflective heuristic outperformed other scheduling schemes in completion time. It demonstrated that it could outperform the second best approach using Max-min heuristic in the test with 38 control points and 10,000 triangles. The completion time decreased by about 39%. It could also outperform the second best Application-oriented heuristic in the test with control 20 points and 10,000 triangles, by about 21% . Therefore, we can say that the reflective design and the skeleton library not only simplify the Grid task design but also significantly improve performance.

7.2 Future work

In the future, several areas can be improved:

- In the library, there are only two types of skeletons available, farm and pipeline. It is possible to add other frequently used skeletons into this library in order to extend the scope.
- The recovery mechanism. The scheduler simply re-schedules the tasks to another computing resource if they cannot be done in time. If there are some task migration functions, a are found in Condor, the performance of the scheduler can be much improved.
- The security problem. Methods of adding authentication functions will be very important in the future, as under current conditions, hostile users could destroy the whole system.
- Stability in static environments. As discussed in Chapter 5, the scheduling heuristic does not perform efficiently in a static environment. The reason for this is partly because of the design of the two embedded heuristics. Therefore, in the future, it will be very important to re-design both the System-oriented heuristic and Application-oriented heuristic to suit both static and dynamic environments.
- The forecasting techniques for the available computing resources for next period. In this thesis, the Blume adjustment is used to predict the available resources for the next period, however it may not be the best forecasting technique. In the future, the Vasicek [16] adjustment and Bayesian

approach can be explored to enhance the forecasting power in some degree. In addition, the stationary test is not enough if we only look at the t-statistical data. In the future, the unit root [16] test should become the key criterion. Also the forecasting criterion should be replaced by using the Mean Square Forecasting Error.

Publications

[1] Ma, Y., Hillston, J., Cole, M, "**A skeleton middleware for the parallel program**." Parallel and Distributed Computing, Applications and Technologies, 2003. PDCATapos;2003. Proceedings of the Fourth International Conference on Volume , Issue , 27-29 Aug. 2003 Page(s): 541 – 547

[2] Chris Thompson, Yuke Ma, Frank Wang, "**Implementation of the reflective scheduler of a Grid**", (Accepted) Journal of Grid Computing

[3] Yuke Ma, Chris Thompson, "**Using welfare economy theory in dynamism management for the Grid System**.",(Accepted) The 2008 International Conference on Grid Computing and Applications, Las Vegas, USA. Proceedings of the 2008 International Conference on Grid Computing and Applications (GCA'08)

[4] Chris Thompson, Yuke Ma, "**Design a reflective scheduler with task partitioning support**" (Waiting for submission)

Reference

[1] A. Reinefeld, R. B., T. Decker, J. Gehring, D. Laforenza, F. Ramme, T. Romke, J. Simon (1997). The MOL project: an open, extensible metacomputer. 6th Heterogeneous Computing Workshop (HCW '97).

[2] Abhijit Bose, B. W., Cameron Wood (2004). MARS: A Metascheduler for Distributed Resources in Campus Grids. Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04).

[3] Abramson, D., Sosic, R., Giddy, J., and Hall, B. (1995). Nimrod: a tool for performing parametrised simulations using distributed workstations. In Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing (HPDC '95), Pentagon City, VA, USA, IEEE CS Press , Los Alamitos, CA, USA.

[4] Abramson, D., Giddy, J., and Kotler, L (2000). High Performance ParametricModeling with Nimrod/G: Killer Application for the Global Grid? . In Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun, Mexico, IEEE CS Press, Los Alamitos, CA, USA.

[5] Acharya, A., Uysal, M., Bennett, R., Mendelson, A., Beynon, M., Hollingsworth, J.,Saltz, J., and Sussman, A. (1996). Tuning the performance of i/o-intensive parallel applications In Proceedings of the fourth workshop on I/O in parallel and distributedsystems (IOPADS '96), Philadelphia, PA, USA,

ACM Press.

[6] Ahmad, Y. K. K. a. I. (1999). Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Computing Surveys.

[7] Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., and Tuecke, S. (2002). "Data management and transfer in high-performance computational grid environments." Parallel Computing 28(5).

[8] Allcock, W. (2003). GridFTP Protocol Specification, Global Grid Forum Recommendation GFD.20**:** 189-190.

[9] Amdahl, X. (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIPS Conference Proceedings.

[10] Anglano, C. (2001). Integrating GRID tools to build a computing resource broker: activities of DataGrid WP1. In Proceedings of the 2001 International Conferenceon Computing in High Energy and Nuclear Physics (CHEP), Beijing, China.

[11] Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., and K. Sagi, Shi, Z., and Vadhiyar, S. (2002). Users' Guide to NetSolve V1.4.1, Knoxville, TN, Innovative Computing Dept. University of Tennessee.

[12] Baker, M., Buyya, R., and Laforenza, D. (2002). Grids and Grid Technologies for Wide-Area Distributed Computing. Software: Practice and Experience, Wiley Press,USA. 32**:** 1437-1466.

[13] Berman, F. a. W., R. (1997). The AppLeS Project: A Status Report. In Proceedings of the 8th NEC Research Symposium, Berlin, Germany.

[14] Berna Massingill, T. M., Beverly Sanders (1994). A Pattern Language for Parallel Application Programming, Wiley Press.

[15] Bester, J., Foster, I., Kesselman, C., Tedesco, J., and Tuecke, S. (1999). GASS: A Data Movement and Access Service for Wide Area Computing Systems. In proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems, Atlanta, USA, ACM Press, New York, NY, USA.

[16] Blume, M. (1975). "Betas and Their Regression Tendencies." Journal of Finance 30: 785-795.

[17] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible Markup Language (XML) 1.0 (3rd Edition).

[18] Buda, G. C., D. Graveman, R.F. Kubic, C. Booz Allen and Hamilton, Linthicum, MD (2001). Communications for Network-Centric Operations: Creating the Information Force. Security standards for the global information grid in: Military Communications Conference, 2001. MILCOM 2001. , IEEE Press.

[19] Buyya, R., Abramson, D., and Giddy, J. (2000). Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China, EEE Computer Society Press,USA.

[20] C.E.Catlett, L. S. a. (1992). Metacomputing. Communications of the ACM.

[21] Casanova, H., Obertelli, G., Berman, F., and Wolski, R. (2000). The AppLeS parameter sweep template: User-level middleware for the grid. In

Proceedings of the 2000 ACM/IEEE conference on Supercomputing (SC'00), Dallas, TX, USA, IEEE Computer Society.

[22] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. (2000). Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In Proceedings of the 9th Heterogeneous Computing Systems Workshop (HCW 2000), Cancun, Mexico, IEEE CS Press, Los Alamitos, CA, USA.

[23] Casanova, H. (2000). Heuristics for Scheduling Parameter Sweep Applications in Grid Environments 9th HCW.

[24] Casanova, H. a. B., F. (2003). Parameter Sweeps on the Grid with APST. Grid Computing, Wiley Press, London, UK.

[25] Catlett, C. (2002). The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).

[26] Chapin, S., Karpovich, J., and Grimshaw, A. (1999). The Legion resource management system. In Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, San Juan, Puerto Rico, IEEE CS Press, Los Alamitos, CA, USA.

[27] Cheney, D. K. a. W. (2005). Numerical Analysis: Mathematics of Scientific Computing, Third Edition. Thomson Learning.

[28] Ciardo, G. M., J. Trivedi, K (1989). SPNP: stochastic Petri net package In Proceedings of the Third International Workshop on Petri Nets and Performance Models.

[29] Cole, M. (2001). Bringing Skeletons out of the Closet, Institute for

Computing System Architecture Division of Informatics, University of Edinburgh.

[30] Czajkowski, K., Foster, I. T., Karonis, N. T., Kesselman, C., Martin, S., Smith, W.,and Tuecke, S. (1998). A ResourceManagement Architecture forMetacomputing Systems. In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing(IPPS/SPDP '98), Orlando, Florida, USA, Springer-Verlag, Berlin, Germany.

[31] Czajkowski, K., Kesselman, C., Fitzgerald, S., and Foster, I. (2001). Grid information services for distributed resource sharing. In Proceedings of the 10th IEEEInternational Symposium on High Performance Distributed Computing (HPDC-10), San Francisco, CA, CS Press, Los Alamitos, CA, USA.

[32] D. Falco, R. D. B., E.Tarantino, and R. Vaccaro (1994). Improving search by incorporating evolution principles in parallel tabu search. In IEEE Conference on Evolutionary Computation.

[33] Deelman, E., Blythe, J., Gil, Y., and Kesselman, C. (2003). Grid Resource Management:State of the Art and Future Trends. Workflow Management in GriPhyN, Kluwer Academic Publishers, Cambridge, MA, USA.

[34] Eckel, B. (2006). Thinking in Java (4th Edition), Prentice Hall PTR; ISBN-10: 0131872486.

[35] Erwin, D. W. a. S., D. F. (2001). UNICORE: A Grid Computing Environment. In Proceedings of the 7th International Euro-Par Conference on ParallelProcessing (Euro-Par '01), Manchester, UK, Springer-Verlag, Berlin, Germany.

[36] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. (1998). A security

architecture for computational grids. In Proceedings of 5th ACM Conference on Computer and Communications Security Conference, San Francisco, CA, USA, ACM Press,New York, NY, USA.

[38] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications.

[39] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002). Grid services for distributed system integration. Computer.

[40] Foster, I., Tuecke, S., and Unger, J. (2003). OGSA Data Services. Global Grid Forum 9.

[41] Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2005). Modeling and managing State in distributed systems: the role of OGSI and WSRF. Proceedings of the IEEE.

[42] Foster, I. a. K., C. (1998). The Globus Project: A Status Report. In Proceedings of IPPS/SPDP'98 Heterogeneous Computing Workshop, Orlando,FL, USA, IEEE CS Press , Los Alamitos, CA, USA.

[43] Foster, I. a. K., C. (1999). The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, San Francisco, USA.

[44] Foster, I. a. K., C. (1999). The Grid: Blueprint for a new computing infrastructure. The Globus toolkit, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[45] Frey, J., Tannenbaum, T., Livny, M., Foster, I., and Tuecke, S. (2002). Condor-G: A Computation Management Agent for Multi-Institutional Grids.

Cluster Computing.

[46] G.S.Blair, G. C., P.Robin, M.Papathomas (1998). An Architecture for Next Generation. Proceedings of SIGCOMM'98.

[47] Gagliardi, F., Jones, B., Reale, M., and Burke, S. (2002). European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications. In Proceedings of Performance Evaluation of Complex Systems: Techniques and Tools,Performance 2002, Springer-Verlag, Berlin, Germany.

[48] Glave, J. (1997). Macho Computing at Root of RSA Contest Flap. Wired.

[49] Gropp, W. L., Ewing; Skjellum, Anthony (1994). Using MPI: portable parallel programming with the message-passing interface. Scientific And Engineering Computation Series, MIT Press,Cambridge, MA, USA ,ISBN 0-262-57104-8.

[51] H. Chen, N. S. F., and D. W. Watson (1998). Parallel genetic simulated annealing: a massively parallel SIMD approach. IEEE Transactions on Parallel and Distributed Computing.

[52] H. J. Siegel, H. G. D., and J. K. Antonio (1997). The Computer Science and Engineering Handbook, . Software support for heterogeneous computing.

[53] James Patton Jones, B. N., Bob Henderson (2001). Workload Management: More Than Just Job Scheduling. 3rd IEEE International Conference on Cluster Computing (CLUSTER'01).

[54] Jensen, K. (1980). High-level Petri nets. In Proceedings of 3rd Eur. Workshop on Appl. and Theory of Petri Nets.

[55] JOSH Project, E. U. (2004). JOSH Project. Available at http://www.epcc.ed.ac.uk/sungrid last visit: 06-Mar-2008.

[56] Kim, O. I. a. C. (197). Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM.

[57] L. Wang, H. J. S., V. P. Roychowdhury, and A. A. Maciejewski (1997). Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. Journal of Parallel and Distributed Computing.

[58] Laguna, F. G. a. M. (1997). Tabu Search, Kluwer Academic Publishers.

[59] Legrand, I. C. a. N., H. B. (2000). The MONARC toolset for simulating large network-distributed processing systems. In Proceedings of the 32nd Winter Simulation Conference (WSC '00), Orlando, FL, Society for Computer Simulation International,San Diego, CA.

[60] Litzkow, M., Livny, M., and Mutka, M. W. (1988). Condor - a hunter of idle workstations. In Proceedings of the 8th Int'l Conference of Distributed Computing Systems, , Los Alamitos, CA, USA, IEEE CS Press.

[61] Liu, K. C. a. B. (1991). On mapping signal processing algorithms to a heterogeneous multiprocessor system. ICASSP 91.

[62] Louridas, P. (2006). SOAP and Web Services. IEEE Software.

[63] M. Maheswaran, S. A., H. J. Siegel, D. Hensgen, and R. Freund (1999). Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In the 8th IEEE Heterogeneous

Computing Workshop(HCW'99).

[64] M. Maheswaran, S. A., H.J.Siegel, D. Hensgen, and R. Freund (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems. 8th Heterogeneous Computing Workshop (HCW'99).

[65] M. Maheswaran, T. D. B., and H. J. Siegel (1999). . Encyclopedia of Electrical and Electronics Engineering. Heterogeneous Distributed Computing, John wiley & sons.

[66] Management, L. R. Available at: http://www.ibm.com/developerworks/linux/library/l-memory/. last visit: 05-Sep-2008.

[67] Middleware (2000). IFIP/ACM International Conference on Distributed Systems Platforms, New York, NY, USA.

[68] Middleware (2001). IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany.

[69] Nakada, H., Sato, M., and Sekiguchi, S. (1999). Design and implementations of Ninf: Towards a global computing infrastructure. Future Generation Computing Systems.

[70] Note, W. S. D. L. W.-W. C. (2001). Available at: http://www.w3.org/TR/wsdl. last visit: 06-Mar-2008.

[71] Palazzari, M. C. a. P. (1996). Real time pipelined system design through simulated annealing. Journal of Systems Architecture.

[72] Pataricza, A. (2004). Formális moódszerek az informatikában (Formal methods in informatics). TYPOTEX Kiadó. ISBN 963-9548-08-1.

[73] Virtual PC. Available at: http://www.microsoft.com/windows/winfamily/virtualpc/default.mspx. last visit: 02-Sep-2008.

[74] Peterson, J. L. (1977). Petri Nets. ACM Computing Surveys

[75] Peterson, J. L. (1981). Petri Net Theory and the Modeling of Systems, Prentice Hall,ISBN 0-13-661983-5

[76] Petri, C. A. Petri net, Scholarpedia.

[77] Petri, C. A. (1962). Kommunikation mit Automaten, Ph. D. Thesis. University of Bonn.

[78] Project, J. (2005). Jabber Protocols. Available at http://www.jabber.org/protocol/, last visit: 06-Mar-2008.

[79] R. Armstrong, D. H., and T. Kidd (1998). The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In 7th IEEE HeterogeneousComputing Workshop (HCW ’98).

[80] R. Freund, M. G., S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust, and H. Siegel (1998). Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In 7th IEEE Heterogeneous Computing Workshop (HCW ’98).

[81] Raman, R., Livny, M., and Solomon, M. (1999). Matchmaking: An

extensible framework for distributed resource management. Cluster Computing,.

[82] Reed, D. A. (2001). Teragrids: The Future of Terascale Cluster Computing. 3rd IEEE International Conference on Cluster Computing (CLUSTER'01).

[83] Reed, D. A. (2003). Grids, the TeraGrid, and Beyond. Computer.

[84] Reed, W. J. The Pareto, Zipf and other power laws. Economics Letters.

[85] Reisig, W. A Primer in Petri Net Design, Springer-Verlag. ISBN 3-540-52044-9.

[86] Riemann, R.-C. Modelling of Concurrent Systems: Structural and Semantical Methods in the High Level Petri Net Calculus, Herbert Utz Verlag. ISBN 3-89675-629-X.

[87] Rizos Sakellariou, H. Z. (2004). A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. In Proceedings of the IEEE Heterogeneous Computing Workshop, IEEE Computer Society Press.

[88] Roskies, R. (1994). Metacomputing – Pipedream or Practical Reality. Computers in Physics.

[89] S. Kirkpatrick, J. C. D. G., and M. P. Vecchi (1983). Optimization by simulated annealing. Science.

[90] Seti@Home. Available at: http://setiathome.berkeley.edu, Last visit: 06-Mar-2008.

[91] Siegel., R. F. F. a. H. J. (1993). Heterogeneous processing. IEEE Computer.

[92] Specification, U. V. P. (2002). Available at: http://uddi.org/pubs/uddi_v3.htm. last visit: 06-Mar-2008.

[93] Störrle, H. Models of Software Architecture - Design and Analysis with UML and Petri-Nets, ISBN 3-8311-1330-0.

[94] T DeFanti, I. F., M Papka, R Stevens, T Kuhfuss (1996). Overview of the I-WAY: Wide area visual supercomputing. Journal of Supercomputer Applications.

[95] T. Braun, H. S., N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B.Yao, D. Hensgen, and R. Freund (1999). A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. 8th IEEE Heterogeneous Computing Workshop (HCW ’99).

[96] Takefusa, A., Tatebe, O., Matsuoka, S., andMorita, Y. (2003). Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In Proceedings of the 12th IEEE international Symposiumon High Performance Distributed Computing (HPDC-12), , Seattle, USA, IEEE CS Press, Los Alamitos, CA, USA.

[97] Tatebe, O., Soda, N., Morita, Y., Matsuoka, S., and Sekiguchi, S. (2004). Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04) Conference, Interlaken, Switzerland.

[98] Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., and Sekiguchi, S. (2002). Grid Datafarm Architecture for Petascale Data Intensive Computing. In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany, IEEE CS Press, Los Alamitos, CA, USA.

[99] Tatebe, O., Ogawa, H., Kodama, Y., Kudoh, T., Sekiguchi, S., Matsuoka, S., Aida,K., Boku, T., Sato,M.,Morita, Y., Kitatsuji, Y.,Williams, J., and Hicks, J. (2004a). The Second Trans-Pacific Grid DatafarmTestbed and Experiments for SC2003. In Proceedings of 2004 International Symposium on Applications and the Internet – Workshops (SAINT 2004 Workshops)), Tokyo, Japan, IEEE CS Press, Los Alamitos, CA, USA.

[100] Thain, D., Bent, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., and Livny, M. (2001). Gathering at the well: Creating communities for grid I/O. In Proceedings of Supercomputing 2001, Denver, Colorado, IEEE CS Press, Los Alamitos, CA, USA.

[101] Thain, D., Basney, J., Son, S.-C., and Livny, M. (2001a). The Kangaroo Approach to Data Movement on the Grid. In Proc. of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, CA, IEEE CS Press, Los Alamitos, CA, USA.

[102] Ullman, J. D. (1975). NP-complete Scheduling Problems. Journal of Computer and System Sciences.

[103] Velev, M. N. (2004). Formal verification of pipelined processors with load-value prediction. Ninth IEEE International High-Level Design Validation and Test Workshop.

[104] Vipin Kumar, A. G., Anshul Gupta and George Karpis (1994).

Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms, Benjamin-Cummings Pub Co, 978-0805331707.

[105] White, B. S., Grimshaw, A. S., and Nguyen-Tuong, A. (2000). Grid-Based File Access: [] The Legion I/O Model. In Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00), Pittsburgh, USA, IEEE CS Press,Los Alamitos, CA, USA.

[106] Wolski, R., Spring, N., and Hayes, J. (1999). The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems.

[107] Xu.Bo (2007). A New Method Using B-splines as Shape Functions and the Knot Insertion Algorithm for Shape Optimization. PhD thesis, Cranfield University.

[108] Yang, L., Foster, I. and Schopf, J.M (2003). Homeostatic and Tendency-based CPU Load Predictions. International Parallel and Distributed Processing Symposium (IPDPS2003), Nice, France.

[109] Youssef, H. S. a. A. (1996). Mapping and scheduling heterogeneous task graphs using genetic algorithms. In 5th IEEE Heterogeneous Computing Workshop (HCW '96).

[110] Z. Xu, B. F., W. Li (2004). Grid Computing Technology. China Electronics Press.

[111] Zhou, M. a. K. V. Modeling, Simulation, & Control of Flexible Manufacturing Systems: A Petri Net Approach. World Scientific ISBN 981-02-3029-X.

[112] Zhou, M. F. D. Petri Net Synthesis for Discrete Event Control of Manufacturing Systems, Kluwer Academic Publishers. ISBN 0-7923-9289-2.

Appendix Task Partitioning Support

In addition to challenges coming from the dynamic infrastructure, the Grid faces another difficulty: task partition. As we know, when the CPU speed is approaching its limit, it becomes very difficult to improve the computational power from just a single computing resource. In this case, people pay more and more attention to parallel computing. To run parallel computing, the pre-

condition is that the task can be divided into sub-tasks. However, the complexity in programming and migrating continuously restricts the development of parallel computing. Until now, parallel computing technologies are not widely implemented because of the above reasons [110]. However, researchers are now used to the sequential programming, so if there is a solution to support the task partition and allocate the sub-task into different nodes to execute, and then parallel programming can make breakthrough progress. It is evident that the Grid provides users with the physical infrastructure to run parallel programs. Because this is convenient, there is an increasing requirement for parallel technologies. So the method of allowing the users to take advantage of the Grid becomes another increasing concern. If the Grid can provide a kind of service which helps the users to divide the task into sub tasks and manage these tasks, then the parallel computing becomes just as easy as traditional sequential computing. The original idea of the skeletal approach comes from parallel programming. It proposes that commonly used patterns of parallel computational interaction should be packaged up as parameterisable library classes (or language control structures) so that subsequent users can benefit from finely tuned implementations of the underlying structure, without "re-inventing the wheel" and without risking the introduction of complex errors. In this PhD thesis, we will investigate the possibility of providing a library based on Java Mulit-thread standard APIs for the Grid task partition. This could involve considering the underlying conceptual model for such an API, designing and implementing

some skeleton classes in this style and working with some sample applications to illustrate and evaluate these. Combining with the APIs from Java Cog, the library now is able to provide the partitioning support and communication with the Grid.

A.1 Skeleton Programming

Nowadays, one of major concerns in parallel programming [14][68] is to decide whether to transplant an existing sequential program or to develop an absolutely new parallel application program. There are three ways to develop parallel applications. The first is based on the automated parallelism mechanism [67][68] and the second is to rewrite these sequential programs. The purpose of automated parallelism is to free users from the onerous work of parallelizing programs. The compiler receives codes with the parallel tags and with little users' input, then can produce highly efficient parallel object codes. But it can be noted that this goes beyond the current compilation technology, in other words, it may be impossible under current technology. The second way is to use the parallel library. In this feasible way, users port the sequential programs by using the parallel library. This way appears to be much more successful than the first one.

Compared to the parallel computing architecture, the Grid can be regarded as a distributed parallel computer. The difference is that the Grid has a loose

structure and the parallel architecture is fixed during the running time. However, some of matured technologies in parallel computing can be transferred to the Grid environment to improve the performance of the Grid, i.e. how to partition tasks. Most of these schedulers implement the second idea and develop their own language to rewrite requests in order to support those running in the Grid environment. Some of them called the workflow based scheduler take advantage of the concept workflow, which is a kind of business processing pattern, not restricted to the fixed business processes. It allows the users to submit their workflow descriptions and map these workflows to the real requests in the Grid.

However, asking the user to provide the Directed Acyclic Graph (DAG) [89] may not be the most suitable thing to do in the Grid environment. Because the Grid environment is very complicated, integrating the heterogeneous systems together, the user may not be able to handle this. Furthermore, given the problem, the user may not provide a highly efficient way to separate problems. In addition, the user still needs to worry about the task partition and devote time to it. Therefore this thesis should explore an alternative way which encapsulates the frequently used skeletal codes into a library implemented by highly efficient codes.

The library [29] previously developed only supports the shared memory systems, so in some ways it is distant from the Grid. To achieve the intended goal, the library needs to be modified to support both heterogeneous and

homogeneous programming environments. The heterogeneous environment includes different operating systems and different network protocols. There are many ways to overcome the differences, such as using message passing and middleware. Java is popular for its adaptability and simplicity. Java programs can be run in a heterogeneous environment.

In the design, the user will not be bothered by the onerous work in constructing the parallelism in the Grid computing environment, for the library provides methods to hide the low level details, to let the user work in a traditional way.

Again, the skeleton model is inspired by the eSkel [29] design. In their design, they abstract the most re-usable skeletons for the parallel program, like Pipeline, Farm and so on, then put the skeleton in the eSkel Library.

The library is made up of three modules:

- User side: this is utilized by the users to demonstrate the task of programs and more importantly for the users to choose a suitable skeleton for their programs.
- Grid side: this hides the low level details and communicates with the Grid. The most important part in this system is the Skeleton Library: this includes the Skeleton Library. The Skeleton Library is used to store all the parallel skeletons, offering the Skeleton Service a match to the logical topology.

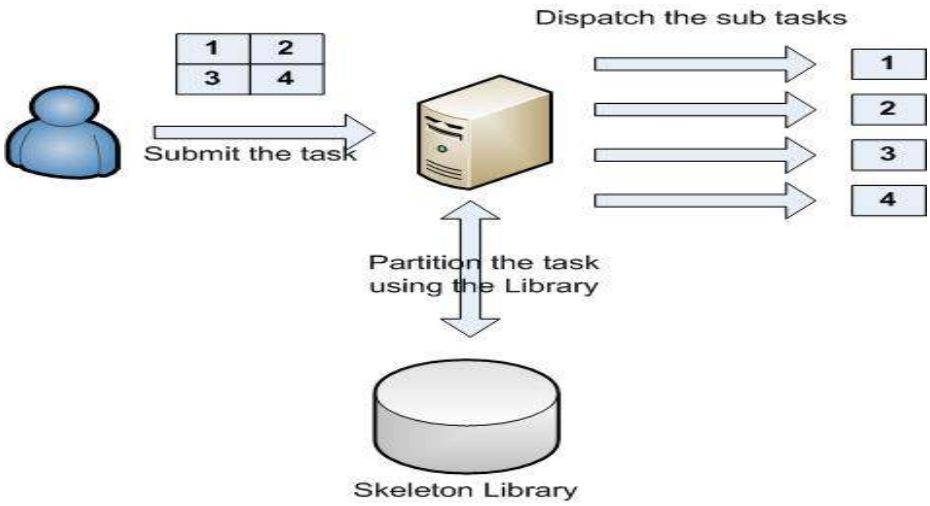

Figure A1: the general process describes how to partition the task using the skeleton library

Here, we design the system from the view of users. When beginning to construct the program, the user must first decide which skeleton to use. Then comes decisions on the parameters used to define skeleton type, the scale of the skeleton etcetera. Finally, the system executes the uploaded codes and data, and returns results to the user. It significantly eases the onerous work for the user to construct the parallelism in the Grid computing environment.

A.2 Overall Design

The library could be reused by other programs. There are two kinds of skeleton library:

- To encapsulate the frequently used control structures, and

- To provide some mathematical methods.

So the structured skeletal programming library proposes that commonly used patterns of skeletal computational interaction should be packaged up as parameterisable library functions (or language control structures) so that subsequent users can benefit from finely tuned implementations of the underlying structure, without "re-inventing the wheel" and without risking the introduction of complex errors. The possibility will be investigated of providing a similar library based around the Java Thread standard API for the Grid programming. This will involve considering the underlying conceptual model for such an API, designing and implementing some skeleton functions in this style and/or working with some sample applications to illustrate and evaluate these.

We can discuss the major advantages of the idea as follows:

- Programmable: A set of existing Grid programming solutions can greatly improve the user's work rate. The core idea is to hide the system's low level details, encourage codes re-use and relieve the burdens of the users. The users can spend more time in modifying the application program itself rather than the details of the low level programming environment.
- Reusable: In software engineering, re-use is a hot subject. The skeleton library will help users to implement re-use. Because this method allows

different programs to use the same parallel control structures, this will also avoid the duplication of work during development.

- Portable: To provide parallel application programs portability is a very important issue. It enables the programs to be run on different system environments.
- Efficient: To improve the performance and improve the portability is a paradox. They both play important roles.

The aim of the design is to allow users to share knowledge about their algorithm design. In Grid, the users meet many control structures that have occurred, and will occur again. Documenting and encapsulating them into a library will help the users to reuse the control structure and possibly relieve the onerous parallelism task. To develop a successful library, we must address the following aspects:

- **The context to which this library applies**.

  The library original is designed for the pure shared memory environment. The entire memory in the system uses the global uniform address. Now the library needs to change in order to suit the Grid.

- **Prerequisites that should be satisfied before using the library**

  Before the user utilizes the library, they must first import the library. Why put all the skeletons in the library? Because the library should make the user believe that the library is reliable even if it releases a new edition. On

one hand, the users do not need to change their codes when the library updates. On the other, the library developers can optimize the library without affecting the existing Grid programs.

- **The library implementation**

  Why use Java to implement the library? People become more and more interested in Java, in fact it is almost the most popular language in recent years. The property aids software development in cluster systems. The need for high performance and powerful computing resources led to great advances in the area of distributed and Grid computing. Currently cluster machines, Grids and heterogeneous networked systems can provide processing power comparable to special purpose multi-processor systems for a fraction of the cost. Java [34] can provide the user with transparent and efficient utilization of multiple machines used in cluster or distributed systems.

- **The criterion of the library**

  The library design could be viewed as a pattern design, for the skeleton is also an essence of the Grid program to describe the controlling structure of the parallelism. So the criterion for a pattern design is also suitable for the skeleton library. In the paper Design Patterns Software Architecture Course Spring 2000[14], Hausi A. Müller writes

  ✓ “Encapsulation and Abstraction”

Each pattern should only encapsulate the problem and the solution in a well-defined domain. The boundary of the domain should be clear and the solution should only focus on the specified problems. Therefore each pattern is able to abstract the problem and solution accurately.

- ✓ **"**Openness and Variability**"**

  Each pattern should be able to provide the scalability and variability. The scalability is allowed the pattern to work together with other patterns to solve a more complicated problem. The variability provides the flexibility in some degree, allowing other patterns to parameterize it.

- ✓ **"**Equilibrium**"**

  The pattern should be in a balanced condition, which provides a degree of flexibility without losing its boundary of the solution space. The pattern can be easily tailored to certain problems, but it cannot be changed too much to keep the nature of the pattern.

- **Examples**

  The examples will help us to understand the usage of the library and to test whether the function is correct or not.

- **Performance Test**

  The test could help us compare the performance between the traditional parallelization techniques and Skeleton library.

A.3 Specific Design

First the concept of skeletons should be clear. It has two very important characters:

- Providing users with a transparent basic structure and
- Not complete, but can bear some parameters: the number of threads, the input data, the output data and the task for each thread.

To provide simple interfaces and hide the inner details will make programs easy to understand and maintain. In other words, the user now can focus on the program algorithm rather than the parallel control structures and interactions.

If we observe the typical structure of parallel programs, we will find most of the programs introduce a fixed model, such as Pipeline, Farm, Divide&conquer and so on. So we can encapsulate them into the library to improve the re-usability and relieve the tasks of the user. In this project, we present two very frequently used skeletons: Farm and Pipeline.

A.3.1 Task Interface

In this design, we must always notice that the library just needs to provide a transparent parallel structure. The computation works should be independent

of the structure, in other words, the computation task should be completed by the user. So here we define an Interface Task**.** Why use interface? Because the interface is a purer abstract class, to construct the class like this: the function's name, the arguments and so on. The interface only contains the form but not implementation. In this interface, we define one function void Operations (Type input, Type output). When users use the library, they should implement the interface to give computation work to each thread or processes in the parallel programs.

When we design the library we cannot know exact details of how the users implement the interface Task, so this design also takes advantage of the concept of Upcasting [34]. That means we do not need to care about the exact classes which implement the Interface Task, but to just take the base class Task because any new class implements Interface Task() inherent from the base Interface Task() and any interface in Task() must be in the new class. This property greatly helps the library developer to optimize the library.

In the function void Operations (Type input, Type output)**,** the output is public static. To define it as public static will save space in the program and make the result throughout the programs.

A.3.2 Type Class

The Type class is the unit of data which exists in the skeleton methods. Constructing such a class gives the user maximum convenience to use their

own data type. In the function above, void Operations (Type input, Type output), we can see there are two arguments: Type input and Type output. Firstly, no matter what happens to the class Type, they will not affect the library. Because Type class is the unit data in the library, the library will not concern any member data inside the Type class. This property gives the users great freedom to define the number of inputs and the number of outputs. Secondly, it will be easier for the library developer to maintain the library.

A.3.3 Farm Skeleton

A.3.3.1 Intent

This farm [104] skeleton is used to "describe concurrent execution of a collection of independent tasks". In this skeleton, the problem can be divided into several totally independent parts and each part can be executed by an independent thread. Each independent thread executes the same computation, so this kind of program is called a SPMD (Single Program Multiple Data) [104].

A.3.3.2 Motivation

The farm skeleton provides a parallel control structure which is able to divide the task into independent sub-tasks. Each sub-task has a function but with different input parameters.

There are a lot of problems which can be described as the farm skeleton liked problems. Optimizing the skeleton will greatly improve the work efficiency and reduce the workload for the parallel users.

For problems which are close to the embarrassingly parallel, they can also be divided into the same independent tasks. But for them, at the beginning we should allocate the data to each task, and at the end of the program, we should gather results from threads and use such results to do the final computation.

The embarrassing parallel problems could be described in a formal way as follows:

```
Problem P;

P=P1&&P2&&P3&&P4….PN;

Solution(P)=SubSolution(P1) && SubSolution(P2) && SubSolution(P3) &&
SubSolution(P4)…. SubSolution (PN);
```

CODE FRAGMENT 1: PSEUDO CODES FOR THE FARM STRUCTURED PROGRAM

The solution can be divided into several sub solutions. Each of the sub solutions is independent and Sub Solution(Pi) does not depend on Sub Solution(Pj). In other words, the problem could be divided into independent sub-problems and each sub problem is solved in parallel.

For problems close to the embarrassing parallel problems, we can describe the problem like this:

Problem P;

Allocation(P);

$P=P_1 \&\& P_2 \&\& P_3 \&\& P_4 \ldots P_N$;

Solution(P)=SubSolution(P_1) && SubSolution(P_2) && SubSolution(P_3) && SubSolution(P_4)…. SubSolution (P_N);

Gather(P);

CODE FRAGMENT 2: PSEUDO CODES FOR THE PROGRAM CLOSE TO FARM STRUCTURE

A.3.3.3 Implementation:

According to the description above, we can use an array to design the skeleton. In the skeleton, the array could be treated as an embarrassingly parallel problem and each element in the array could be treated as a sub problem. So in the skeleton, a thread repeatedly gets one element synchronously from the input array, and then does the computation according to the users' implementation, finally returning the result to the output array. The idea could be depicted in the Figure A2.

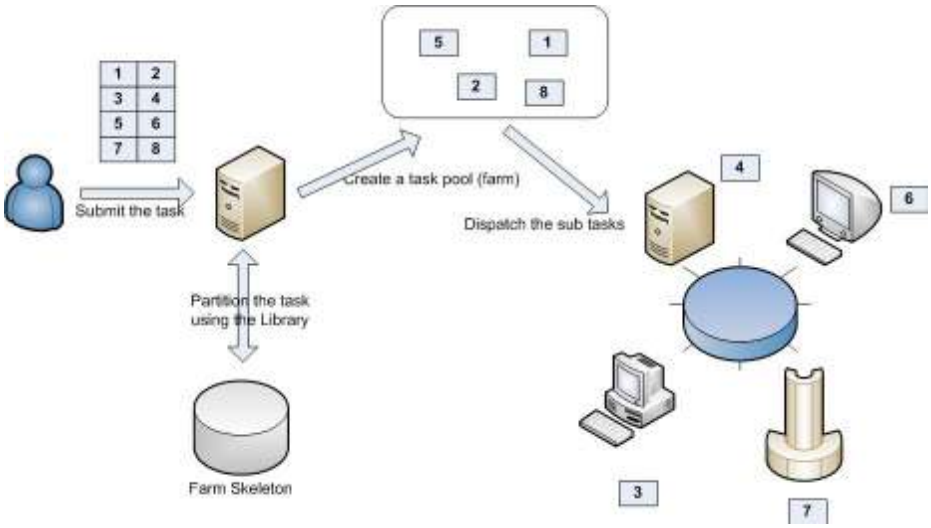

Figure A2: The general process describes how the farm skeleton works in the Grid environment

When we design the farm skeleton, we must consider the following parameters which will affect the controlling structure and should be decided by the user: the number of the threads, the input data, the output data and the computational task for each of the workers in the farm. So we design the farm skeleton Taskfarm Class like this:

```
public Taskfarm(int number,Type[] data,Type[] data2,Task simpletask)
{threads_quit=0;
number_of_threads=number;
array = data;
output =data2;
task=simpletask;
startThreads();}
```

CODE FRAGMENT 3: JAVA TASKFARM CLASS IS USED TO CONSTRUCT THE FARM SKELETON

The constructor function has four parameters: the number of the threads, the input data, the output data and the computational task for each of the workers in the farm. And when a new Taskfarm class is created, the threads will commence by the use of the function startThreads();

Because the input data is an array, the thread repeatedly gets an index of the array. But the input data is shared and there is a possibility for more than one thread to access the same element in the array. So we declare the function getNextIndex() used by each thread to get the index of the input data as synchronized. Each object contains a lock or monitor and this property is “natural” (can be possessed without writing specific codes). When the program calls for any synchronized function, the object is locked and the other synchronized functions of this object cannot access this object until the first function finishes and unlocks the object.

```
public synchronized int getNextIndex()
 {
if(index < array.length) return (index++);else
return(-1);
}
```

CODE FRAGMENT 4: JAVA CODES TO COORDINATE THREADS

While the thread runs, it first gets one element from the input array, then calls for the function Operation() to execute the data. We must notice that the thread here just calls for this function, but how the function works is totally implemented by the users. It is a core part of the skeleton. This approach separates the parallel control structure away from the specific implementation of the task of each thread. The user just needs to focus on improving the efficiency of the computation task. Library developers need to focus on improving the efficiency of the parallel control structure.

Implementations of this pattern include the following key elements:

- How to allocate each of the tasks to each thread. We should notice that the implementation of the task should be independent to the farm skeleton.
- How the problem should be completed and when the problem is completed.

Figure A3 describes the procedure of how the program works.

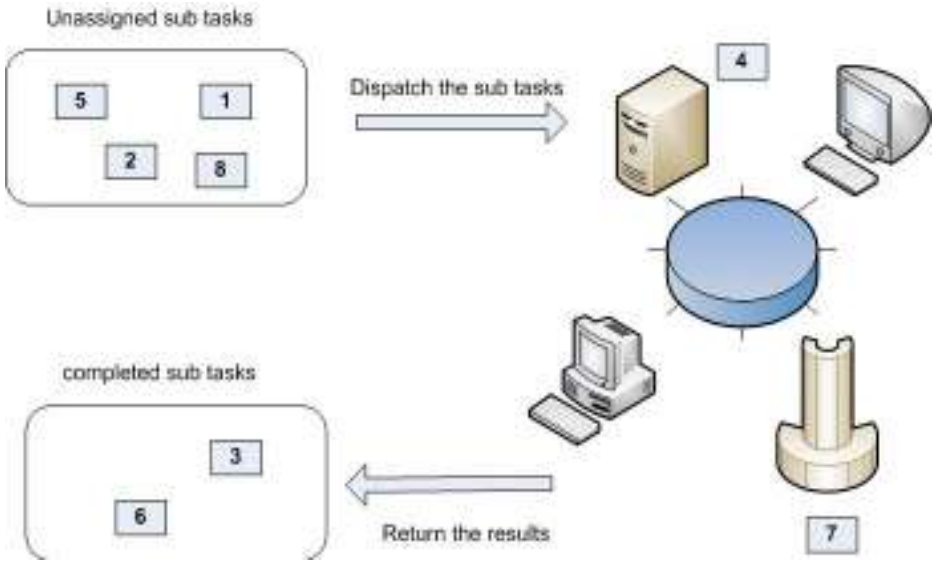

Figure A3: the general process shows how the tasks dispatched to resources and returned when completed

A.3.4 Pipeline Skeleton

A.3.4.1 Intent

Pipeline parallel processing technology is suitable for all the programs which are partial serial programs. In other words, such a problem can be solved by a series of steps. The pipeline structure can be used to parallelize the sequential codes.

A.3.4.2 Motivation

In pipeline technology, the problem should be divided into a serial of ordered sub problems. In fact, this is the principle of sequential programming. In the pipeline, each sub problem is solved by an independent thread and each thread could be regarded as one pipeline step. Every step just completes part

of the problem and leaves the rest of the problem to the next step. Such parallelism could be treated as function decomposition.

We can also describe the pipeline model in a format way:

Problem P;

P= $P_1P_2P_3P_4….P_M$;

Solution(P)=$SubSolution_1(P_1P_2P_3P_4….P_M)$ $SubSolution_2(P_2P_3P_4….P_M)$ $SubSolution_3(P_3 P_4….P_M)$ $SubSolution_4(P_4….P_M)$…. $SubSolution_M(P_M)$

CODE FRAGMENT 5: THE PSEUDO CODES TO DESCRIBE PIPELINE PROGRAMS

In this description, each sub solution has the input stream from the previous step and output stream to the next step except the first sub solution, which only has an output stream, and the last sub solution which only has an input stream.

A.3.4.3 Applicability

If the problem can be divided into a series of ordered tasks, it can be speeded up by using the pipeline parallelism. There are three kinds of computation as follows:

1. To execute multiple instances of a problem.
2. To process a series of data, each item of the data needs to be processed more than once.
3. If one process could forward the all the information needed to the next process before it finishes its own inner operations.

The first kind could be widely used in hardware design simulations and also in experiments which need different parameters to execute several simulations. The second kind, Pipeline, is often used in mathematical computations.

A.3.4.4 Implementation:

Similar to the farm skeleton the pipeline skeleton consists of two classes: the main class Pipeline and the thread class Step. Compared with the MPI [49] programming, the main class can be treated as the master process, and the thread class as the slave process. The main class here is used to receive input data, provide the methods to control each thread, such as the sequence of how the threads do the computation, and return the result back to the main program. The main class is a controlling center. The first problem we should consider is how to control the thread and make them work as Pipeline. First we design each thread as a step in the pipeline structure, that means we allocate one step to one thread and the number of threads is the same as the number of the steps.

Thus we can get to know that each element in the input array should be processed sequentially and each thread can only compute the elements after the previous thread has finished. The available number of tasks for the present step is decided by itself and the previous step.

According to the description, we can use an array to describe the whole problem and each element is a sub task of the problem. The sequence of

elements in the array is also the sequence of the tasks for the whole problem. The input data is an array and each thread computes the elements in the array indexed from 0 to maximum. In each thread, there is a local variable named index to record the next available element which will be computed. All the elements' indices less than index have been computed already. Then we create a public array Number_of_Work here to record the available number of tasks for the thread. To initialize the array at the beginning, the first element in the array which records the first thread's available number of work should be set as the length of the input array and the element left should be set as zero. Because the first step computes all the elements in the input array, the next step can only wait. For example, the second step should wait for the first to finish the first element, and then can start to work. This method can guarantee that all the threads work as a pipeline. Making the array public ensures that all the threads can access this array properly. Figure A4 describes the implementation of the pipeline skeleton.

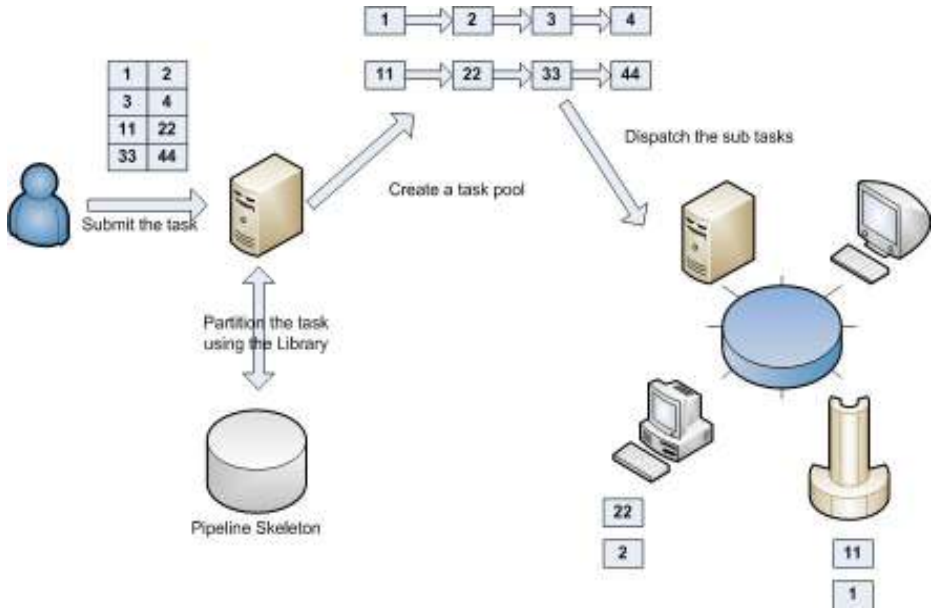

Figure A4: The general process describes how the pipeline skeleton works in the Grid environment

There are two functions to deal with the array:

```
public synchronized int isAvailable(int number)

 {

        if(Number_of_Work[number] >0) return (1);

        else

     return(0);}

 //this function is used for each thread to set the number of available
operations

 //and set the finished condition for all the thread

 public synchronized void setAvailable(int number)

 {Number_of_Work[number]--;

 if(number!=number_of_threads-1)

   Number_of_Work[number+1]++;
}
```

CODE FRAGMENT 6: THE JAVA CODES FOR THE PIPELINE SKELETON TO CONTROL THE STRUCTURE

This first function public synchronized int isAvailable(int number) lets the thread know whether there are any available tasks left. The input argument is the index of the thread. When the function gets the index, it will check the record on the Number_of_Work array. If there is some work left, this function will return one, otherwise zero.

The second function public synchronized int setAvailable(int number) is used to set the number of the available tasks for each thread. The input argument is the index of the thread. When the function gets the index, it will first get the appropriate element in the Number_of_Work array. Because this function is used after the thread finishes one computation, the available work for this thread should decrease by one. Consequently, the available work for the next thread should increase by one, except for the last stage.

The function run() is the core part of the thread class. In this function, we put the threads into two types, one is the final step and others are non-final stages.

If the thread is not the final stage,

```
public void run()

  {int index=0;

    //if the stage is not the final one,the output is the input,

  while(parent.finished!=-1){

    if(number!=parent.number_of_threads-1&&index<=parent.array.length-1)

    parent.task[number].Operations(parent.array[index],parent.array[index++]);

  else if(number== parent.number_of_threads-
1&&index<=parent.array.length-1)


{parent.task[number].Operations(parent.array[index],parent.output[index++]);

if(index==parent.array.length)

 {parent.finished=-1;

}

}

  parent.setNextIndex(number);

}

}
```

CODE FRAGMENT 7: JAVA CODES THAT EXECUTE PIPELINE TASKS

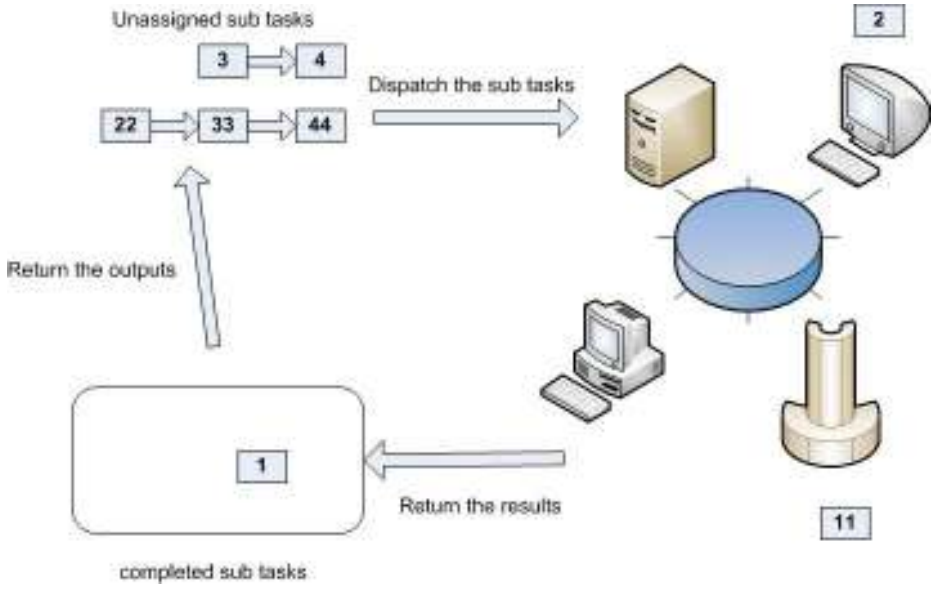

Figure A5: the general process shows how the tasks dispatched to resources and returned when completed

A.4. Experiment

A.4.1 Testing Environment

In order to test the performance of the Skeleton library, we execute two classic examples based on the University of Edinburgh High Performance Computing (HPC) Service.

The cluster consists of 52 900 MHz Ultrasparc III processors in a single cabinet. Each processor has 1 Gb of memory associated with it. The level 1 cache on the UltraSPARC-III is 64kbyte, 4-way set-associative with 32 byte lines.

The University of Edinburgh HPC Service consists of a single SMP cluster configured as follows:

lomond

- The front end (lomond) uses 4 of these processors
- The nominal peak performance of the front end is 7.2 Gflops (900MHz x 4 processors x 2 flops per cycle).
- The level 2 cache on the UltraSPARC-III is 8Mbyte, direct-mapped with 64 byte lines.

The back end

The remaining 48 UltraSPARC-III processors constitute the back end of the HPC service. This has 48 Gbyte of shared memory.

The nominal peak performance of the back end is 86.4 Gflops (900MHz x 48 processors x 2 flops per cycle).

The Grid is constructed with 24 personal computers:

- Intel® Pentium® 4 Processor 2.0G
- 1Gb RAM
- 80GB 7200rpm IDE Hard Disk
- 100MB Motherboard with onboard LAN adapter
- Globus Toolkit 4.0
- Microsoft Windows OEM operating system

Later in this thesis, the performance of the skeleton library will be examined by the Amdahl argument. The Amdahl Argument [9], known as Amdahl's Law, is used to evaluate the overall improvement in the system if some parts of the system improved. This is also an important criterion used to evaluate the performance of the parallelization:

$$\frac{W_s + W_p}{W_s + \frac{W_p}{p}}$$

Where W_s stands for the fixed time needed for parallelization and sequential, W_p stands for the time that can be parallelized and the p stands for the parallelization routines.

A.4.2 Examples

A.4.2.1 Monte Carlo Method Using Farm Skeleton

This skeleton provides a high level programming for such problems. It provides a meta-programming method that lets the user add parameters to the meta-programs to solve the problem.

The principle of the Monte Carlo Method is to use randomization in computation in order to solve numerical physics problems. Because each of the computations is independent, the problem can be classed as embarrassingly parallel computation. (The name Monte Carlo came from the

Capital of Monaco because Monte Carlo is the gambling center and the probabilistic simulation is similar to the winning chance in gambling)

A typical example of the Monte Carlo Method is to calculate π. In the square there is an inscribed circle. The radius of the circle is 1 and the length of the border of the square is 2, so we can get the following equality:

$$\int_0^1 \sqrt{1-x^2}\,dx = \frac{\pi}{4}$$

(This result is suitable for any size of square and its inscribed circle.) So we can choose the points probabilistically from the square and record the number of the points which are also in the square. If we choose enough example points, we can get the proportion of the points in the circle and all the points to be $\frac{\pi}{4}$. In the experiment, we set the image size as size 256 X 256 and the maximum iteration time is 100,000. The experiment result is as follows:

CPU TYPE	NUMBER OF CPU	TIME COST(s)	AMDAHL ARGUMENT
UltraSPARC-III 900MHz	1	371	**1**
UltraSPARC-III 900MHz	2	238	**1.558824**
UltraSPARC-III 900MHz	4	143	**2.594406**
UltraSPARC-III 900MHz	8	95	**3.905263**
UltraSPARC-III 900MHz	10	88	**4.215909**
UltraSPARC-III 900MHz	12	70	**5.3**
UltraSPARC-III 900MHz	24	59	**6.288136**
	NUMBER OF CPU	**TIME COST(s)**	**AMDAHL ARGUMENT**
PIV 2.0G	1	329	**1**
PIV 2.0G	2	315	**1.044444**
PIV 2.0G	4	241	**1.365145**
PIV 2.0G	8	157	**2.095541**
PIV 2.0G	10	130	**2.530769**
PIV 2.0G	12	85	**3.870588**
PIV 2.0G	**24**	**69**	**4.768116**

TABLE 9: THE PERFORMANCE COMPARISON BETWEEN THE CLUSTER AND DISTRIBUTED COMPUTING SYSTEM

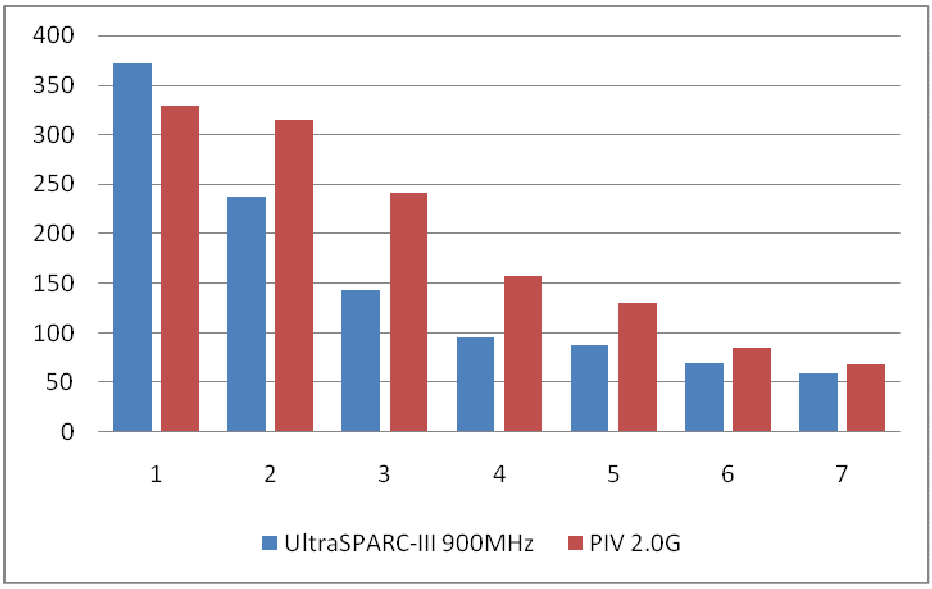

Figure A6: The time costs by using the MPI running on the cluster and the Skeleton library running on Grid, which reflect the efficiency comparison of the Grid and the cluster

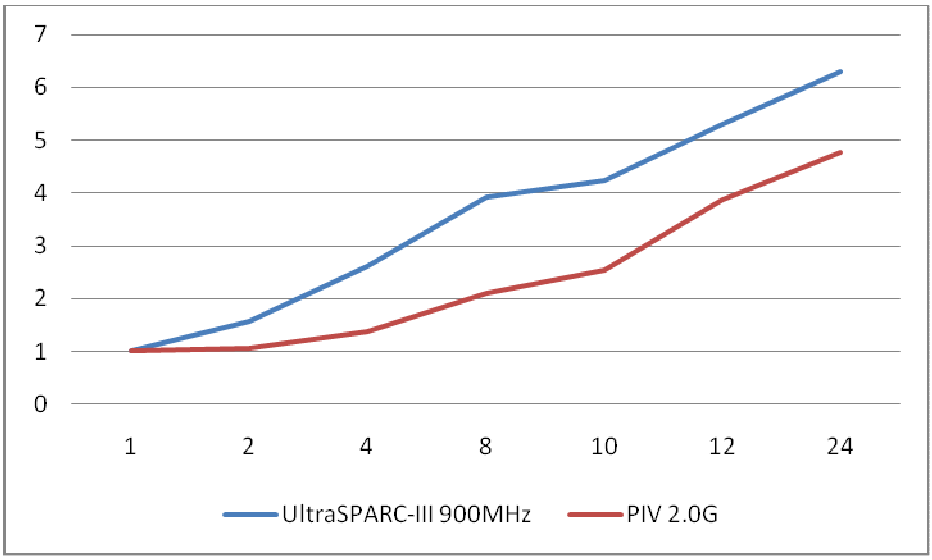

Figure A7: The Amdahl Argument using the MPI running on the cluster and the Skeleton library running on Grid, which reflect the speedup comparison of the Grid and the cluster

From above figures, results suggest that Farm skeleton works efficiently as the number of computing resources increase. The reason is that the cost of the communication is too high and cannot be offset by the parallelization. The more computing resource the Grid uses, the smaller the impact of the overhead.

A.4.2.2 Knapsack Problem

This kind of skeleton is useful in many situations. In sorting problems, the purpose is to sort a series of numbers in increasing order or in decreasing order. In Pipeline, we let the first stage receive one number each time, save the largest number which has been received and send the rest of the numbers to next stages. Each stage executes the same tasks and when the program finishes the first stage has the largest, the second stage has the second largest and so on. In the matrix multiplication and linear equation solution, we can also use the Pipeline to parallelize such a problem.

The knapsack problem is described below. The problem is first represented as an n*c matrix computation and then each thread is allowed to process a chunk set as 2*2 of the matrix at one time.

The calculation of the function F[i,j] follows the formula:

$$F[i,x] = \begin{cases} 0 & ,x \geq 0,\ i = 0 \\ -\infty & ,x < 0,\ i = 0 \\ max_{1 \leq i < n}\{F[i-1,x], (F[i-1, x-w_i] + p_i)\} & ,otherwise \end{cases}$$

The parallelism of the problem could be like this in Figure A8:

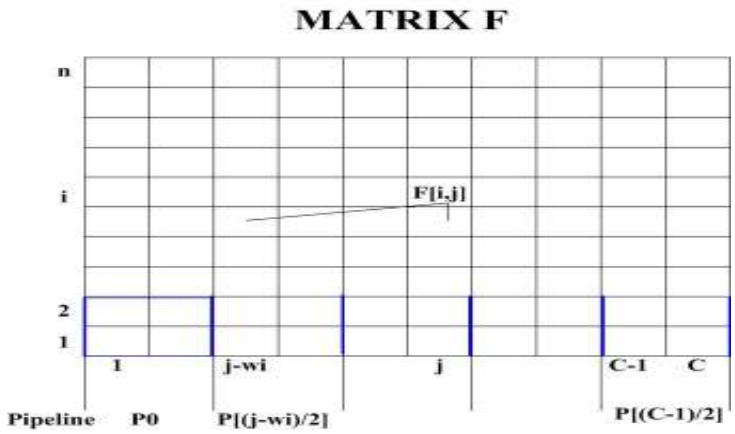

Figure A8: Using the matrix to represent the parallelization of the Knapsack problem

In this experiment, we use

- 32 data: 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10, 40, 30, 50, 10
- the weight of each data is :2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5, 2, 5, 10, 5
- the maximum weight is: 128

CPU TYPE	NUMBER OF CPU	TIME COST(s)	AMDAHL ARGUMENT
UltraSPARC-III 900MHz	1	167	1
UltraSPARC-III 900MHz	2	91	1.835165
UltraSPARC-III 900MHz	4	52	3.211538
UltraSPARC-III 900MHz	8	40	4.175
UltraSPARC-III 900MHz	10	31	5.387097
UltraSPARC-III 900MHz	12	29	5.758621
UltraSPARC-III 900MHz	24	20	8.35
CPU TYPE	NUMBER OF CPU	TIME COST(s)	AMDAHL ARGUMENT
PIV 2.0G	1	96	1
PIV 2.0G	2	103	1.213592
PIV 2.0G	4	87	1.436782
PIV 2.0G	8	65	1.923077
PIV 2.0G	10	48	2.604167
PIV 2.0G	12	35	4.62963
P IV 2.0G	24	19	6.578947

TABLE 10: THE PERFORMANCE COMPARISON BETWEEN THE CLUSTER AND DISTRIBUTED COMPUTING SYSTEM

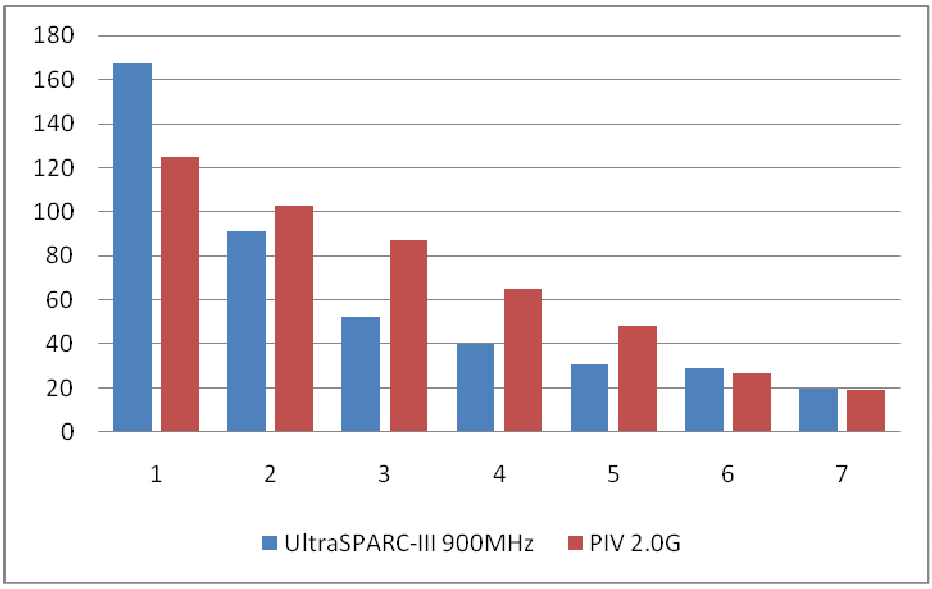

Figure A9: The time costs by using the MPI running on the cluster and the Skeleton library running on Grid, which reflect the speedup comparison of the Grid and the cluster

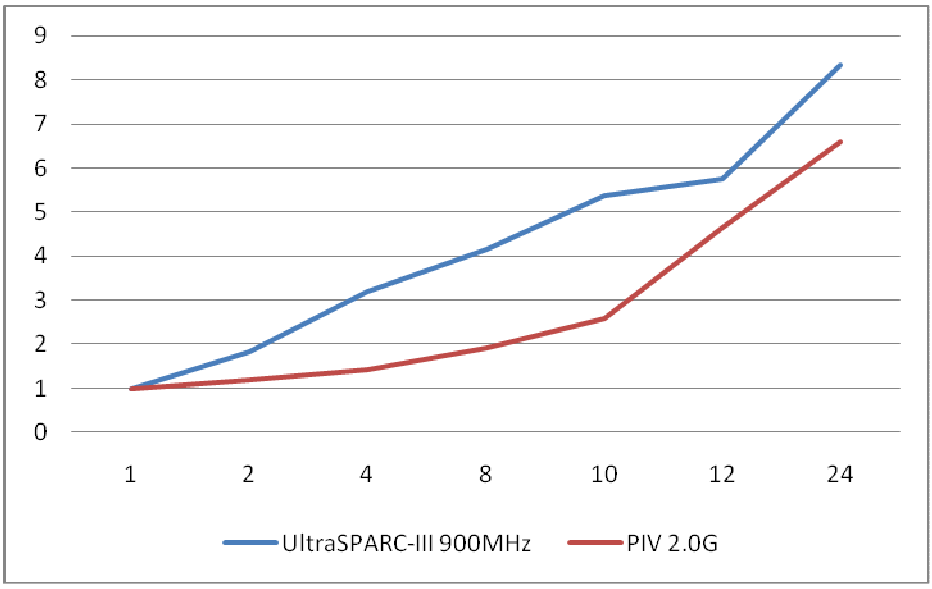

Figure A10: The Amdahl Argument using the MPI running on the cluster and the Skeleton library running on Grid, which reflect the speedup comparison of the Grid and the cluster

From above figures, results suggest that Pipeline skeleton works efficiently as the number of computing resources increase. The reason is that the cost of the communication is too high and cannot be offset by the parallelization. The more computing resource the Grid uses, the smaller the impact of the overhead.

A.4.4 Discussion

From above data, we should notice that the sequential program runs faster than the program using the skeleton library when there is only one computing resource. Why do such results emerge? From the previous introduction, we know that each thread runs on a separate context. In addition to the overhead from the communication with Grid gatekeepers, each thread owns an independent program counter, stack and register and all the threads share the rest of the resources of the programs.

Another thing that should also be kept in mind is the shared resource. In our design, we cannot avoid using a shared resource. To use the lock mechanism to achieve the synchronization, we should pay at least four times the cost of an ordinary function call.

A.5 Summary

In this chapter, the thesis first reviews advantages of using the skeletal programming technique in traditional parallel computers. Then it proposes a way to take advantage of such a technique in the Grid environment. After this, the thesis implements two skeletons: Farm and Pipeline. Finally, a test is implemented to compare the performance of the library in a Grid environment and the MPI in a cluster environment. In the Farm test, when the number of the computing resources is small, say up to 8, the speedup, measured by Amdahl Argument in the cluster almost twice as much as the Grid. When the number of the computing resources is increased to 24, the speedup in the cluster is about 20.8% better than in the Grid. Again in the Pipeline test, when the number of the computing resources is small, say up to 10, the speedup in the cluster is almost twice as much as the Grid. When the number of computing resources is increased to 24, the speedup in the cluster is about 15% better than in the Grid. The reason for this is that the cost of the communication is too high and cannot be offset by the parallelization. The more computing resource the Grid uses, the smaller the impact of the overhead is. When the number of computers in the Grid is big enough, say up to 20, the performance of these two are very close. Therefore, we can say that the skeleton library can provide an efficient and convenient way for the user to partition the tasks.